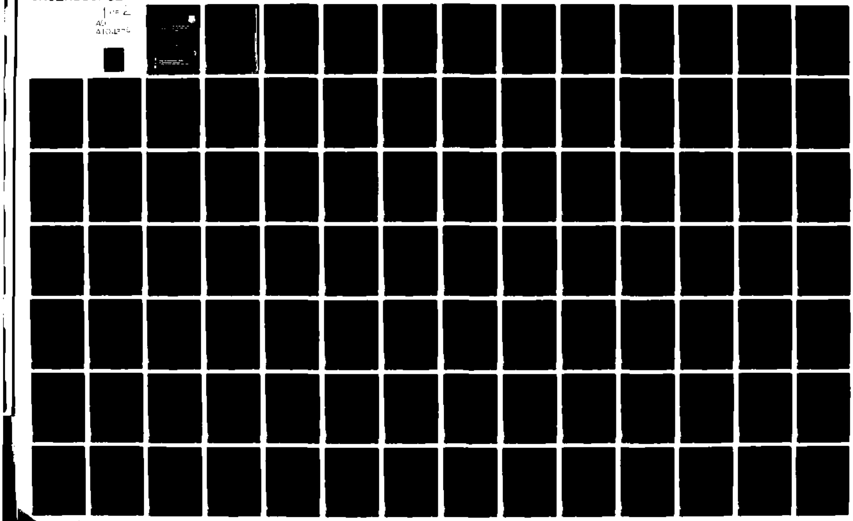
AD-A104 379 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
DISTRIBUTED AND DECENTRALIZED CONTROL IN FULLY DISTRIBUTED PROC--ETC(U)
JUN 81 P H ENSLOW, T G SAPONAS F30602-78-C-0120
UNCLASSIFIED GIT-ICS-81/02 RADC-TR-81-127 NL

1 OF 2
AD
A104379



RADC-TR-81-127
Final Technical Report
June 1981

DISTRIBUTED AND DECENTRALIZED CONTROL IN FULLY DISTRIBUTED PROCESSING SYSTEMS

Georgia Institute of Technology

Philip H. Enslow, Jr.
Timothy G. Saponas

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-127 has been reviewed and is approved for publication.

APPROVED:

THOMAS F. LAWRENCE
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISCP) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-81-127 ✓	2. GOVT ACCESSION NO. AD-A104 379	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DISTRIBUTED AND DECENTRALIZED CONTROL IN FULLY DISTRIBUTED PROCESSING SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 15 Jan 80 – 30 Sep 80
		6. PERFORMING ORG. REPORT NUMBER GIT-ICS-81/02
7. AUTHOR(s) Philip H./Enslow, Jr. Timothy G./Saponas		8. CONTRACT OR GRANT NUMBER(s) F30602-78-C-0120
9. PERFORMING ORGANIZATION NAME AND ADDRESS Georgia Institute of Technology School of Information and Computer Science Atlanta GA 30332		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 31011G R24401P1
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISCP) Griffiss AFB NY 13441		12. REPORT DATE June 1981
		13. NUMBER OF PAGES 116
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Thomas F. Lawrence (ISCP) Supersedes AD-A096 683

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Control
Decentralized Control
Distributed Processing
Fully Distributed Processing Systems
Network
Network Operating System

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Parallel processing has been a popular approach to improving system performance through several generations of computer systems design. Although it is not usually characterized as a "parallel" processing system, a distributed processing system has the inherent capability for highly parallel operation. In order to capitalize on the potential performance improvements achievable by a distributed system, major parallel control problems must be solved. Central to the issue of parallel—

control is the design and implementation of distributed and decentralized control. The study of distributed and decentralized control was initiated with a survey of applicable control models. The results of this survey are presented along with an extensive discussion of the control problems applicable to distributed systems --- specifically "fully" distributed systems.

PREFACE

Comments from the Principal Investigator

Although this is the final report on only one of the approximately 30 research projects currently being performed in the Georgia Tech research program on Fully Distributed Processing Systems, it serves a much broader function than just reporting on the work done in this single project. Since this is the first major technical report published under the program, it has been necessary to document here much of the background applying to the program in general. Specifically, this report presents an extensive discussion of the general philosophies of fully distributed control and fully distributed processing as well as the notation that has been developed to describe the control actions supporting such processing activities.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

BACKGROUND

1.1 GOALS OF COMPUTER SYSTEM DEVELOPMENT

Although the state of the art in digital computers has certainly been advancing faster than any other technologicaι area in history, it is somewhat remarkable that the goals motivating most computer system development projects have remained basically unchanged since the earliest days. Perhaps the most important of these long sought-after improvements are the following:

1. Increased system productivity
   - Greater capacity
   - Shorter response time
   - Increased throughput
2. Improved reliability and availability
3. Ease of system expansion and enhancement
4. Graceful growth and degradation
5. Improved ability to share system resources

The "final or ultimate values" for these various goals cannot be expressed in absolute numbers, so it is not surprising that they continue to apply even though phenomenal advances have been made in many of them such as speed, capacity, and reliability. What is perhaps more noteworthy and important to the discussion being presented here is how little progress has been made in areas such as easy modular growth, availability, adaptability, etc.

It seems that each new major systems concept or development (e.g., multiprogramming, multiprocessing, networking, etc.) has been presented as "the answer" to achieving all of the goals listed above plus many others. "Distributed processing" is no exception to this rule. In fact, many salesmen have dusted off their old lists of benefits and are marketing today's distributed systems as the means to achieve all of them. Table 1 lists some of the benefits currently being claimed for distributed processing systems in current sales literature. Although some forms of distributed processing appear to offer great promise as a possible means to make significant advances in many of the areas listed, the state-of-the-art, particularly in system control software, is far from being able to deliver even a significant proportion of these benefits today.

Table 1. "Benefits" Provided by Distributed Processing Systems

A Representative List Assembled from Claims Made in Actual Sales Literature

High Availability and Reliability

Reduced Network Costs

High System Performance

Fast Response Time

High Throughput

Graceful Degradation, Fail-soft

Ease of Modular and Incremental Growth

Configuration Flexibility

Automatic Load and Resource Sharing

Easily Adaptable to Changes in Workload

Incremental Replacement and/or Upgrade

Easy Expansion in Capacity and/or Function

Good Response to Temporary Overloads

1.2 APPROACHES TO IMPROVING SYSTEM PERFORMANCE

Efforts to improve the performance of digital computer systems can address or be focused on a number of major levels or design issues within the overall computer structure. These levels are:

1. **Materials** - the basic materials used in the construction of operating devices such as transistors, integrated circuits, or other switching devices.
2. **Devices** - operating devices such as transistors, integrated circuits, junctions, etc.
3. **Switching circuits** - design of circuits that provide fast and reliable logic operations.
4. **Register-transfer** - assemblies such as registers, buses, shift registers, adders, etc.
5. **System architecture** - algorithms for executing the basic functions such as arithmetic and logic operations, interrupt mechanisms, control of processor and memory states, etc.
6. **System organization** - the interconnection of major functional units such as control, memory, I/O, arithmetic/logic units, etc., and the rules governing the flow of data and control signals between these units. This level also considers the implementation of multiple, parallel paths for simultaneous operations and transfers.
7. **Network organization** - the number, characteristics, and topology of the interconnection of "complete" systems and the rules governing the control and utilization of the resources those systems provide.
8. **System software** - control and support software for the effective management and utilization of the hardware capabilities provided.

From the very beginning of the computer era there has been activity at all of these levels and such work continues today. (To place it into proper perspective, it should be noted that the research work carried on under this project is focused primarily at the three highest levels, system organization, network organization, and system software, with some work at level 5, system architecture.)

1.3 PARALLEL PROCESSING SYSTEMS

An important theme of computer system development work at levels 5-8, "system architecture," "system organization," "network organization," and "system software," has been parallel processing. Parallel processing has been implemented utilizing approaches focused primarily on the system hardware or the software as well as integrated systems design.

Since the early days of computing, a direction of research that has offered high promise and attracted much attention is "parallel computing." Work in this area dates from the late 1950's which saw the development of the PILOT system [Lein58] at the National Bureau of Standards. The PILOT system consisted of "three independently operating computers that could work in cooperation."[Ensl74] (From the information available, it appears that PILOT would be classified as a "loosely-coupled system" today.) It is interesting to note that the evolution of parallel "hardware" systems lead primarily to the development of tightly-coupled systems such as the Burroughs B-825 and B-5000, the earliest examples of the classical multiprocessor. Other development paths saw the introduction of specialized hardware systems such as SOLOMON and the ILLIAC IV, examples of other forms of tightly-coupled processors.

1.3.1 System Coupling

System coupling refers to the means by which two or more computer systems exchange information. It refers to both the physical transfer of such data as well as the manner in which the recipient of the data responds to its contents. These two aspects of system interconnection are called "physical coupling" and "logical coupling," and they are present in all multiple component systems whether the components of interest are complete computers or some smaller assembly.

The terms, "tight" and "loose" have been utilized to describe the mode of operation of each type of coupling. (Some authors have utilized a third category "medium coupling" and related it to a range of data transfer speeds; however, history has clearly shown that basing any characterizations of digital computers on speed, size, or even cost is an incorrect approach.) The interconnection and interaction of two computer systems can then be described by specifying the nature of its physical coupling and the nature of its logical coupling. It is important to point out that all four combinations of these characteristics are possible and that they all have been observed in implemented systems.

1.3.1.1 Tightly-Coupled Computer Systems

During the 1960's and 1970's, activities in the development of parallel computing, specifically multiple computer systems, were focused primarily on the development of tightly-coupled systems. These tightly-coupled systems took the form of classical multiprocessors (i.e., shared main memory) as well as specialized computation systems such as vector and array processors. This tight physical coupling resulted in a sharing of the directly executable address space common to both processors. There was no means by which the recipient of the data or information being transferred could refuse to physically accept it --- it was already there in his address space.

These early systems also usually implemented tight logical coupling. In this form of system interaction, the recipient of a message is required to perform whatever service is specified therein. With tight logical coupling, there is no independence of decision allowed regarding the performance of the service or activity "requested." The relationship between the sender and recipient is basically that of master-slave.

Although the concept of tightly-coupled multiprocessor systems appears to be a viable approach for achieving almost unlimited improvements in performance (i.e., increases in system throughput) with the addition of more processors, such has not been the results obtained with implemented systems. It is the very nature of tight-coupling that results in limitations on the improvements achievable. Some of the ways that these limitations have manifested themselves are listed below.

1. The direct sharing of resources (memory and input/output primarily) often results in access conflicts and delays in obtaining use of the shared resource.
2. User programming languages that support the effective utilization of tightly-coupled systems have not been adequately developed. The programmer must still be directly involved in job and task partitioning and the assignment of resources.
3. The development of "optimal" schedules for the utilization of the processors is very difficult except in trivial or static situations. Also, the inability to maintain perfect synchronization between all processors often invalidates an "optimal" schedule soon after it has been prepared.
4. Any inefficiencies present in the operating system appear to be greatly exaggerated in a tightly-coupled system.

There was also significant activity during these earlier periods in the development of multiple computer systems characterized as "attached support processors (ASP)." These systems were physically loosely-coupled; but, logically, they were tightly-coupled. The earliest examples of this type of system organization were the use of attached processors dedicated to input/output operations in large-scale batch processing systems. In the latter part of the 1970's, specialized vector and array processors as well as other special-purpose units such as fast Fourier transform units were being connected to general computational systems and utilized as attached support processors. In any event, the specialized nature of the services provided by the attached processor excludes them from consideration as possible approaches to providing general-purpose computational support such as that available from tightly-coupled general-purpose processors functioning as multiprocessors.

Tightly-coupled systems certainly do have a role to play in the total spectrum of computer systems organization; however, their limitations should certainly be considered. It was the recognition of these limitations and the small amount of progress made in overcoming them despite the expenditure of very large research efforts that contributed to the decision to focus our current research program on loosely-coupled systems.

1.3.1.2 Loosely-Coupled Systems

Loosely-coupled systems are multiple computer systems in which the individual processors both communicate physically and interact logically with one another at the "input/output level." There is no direct sharing of primary memory, although, there may be sharing of an on-line storage device such as a disk in the interconnecting input/output communication path. The important characteristic of this type of system organization and operation is that all data transfer operations between the two component systems are performed as input/output operations. The unit of data transferred is whatever is permissible on the particular input/output path being utilized; and, in order to complete a transfer, the active cooperation of both processors is required (i.e., one might execute a READ operation in order to accommodate or accept another's WRITE).

Probably the most important characteristic of loose logical coupling is that one processor does not have the capability or authority to "force" another processor to do something. One processor can "deliver" data to another; however, even if that data is a request (or a "demand") for a service to be performed, the receiving processor, theoretically, has the full and autonomous rights to refuse to execute that request. The reaction of processors to such requests for service is established by the operating system rules of the receiving processor, not by the transmitter. This allows the recipient of a request to take into consideration "local" conditions in making the decision as to what actions to take. It is important to note that it is possible for a system to be physically loosely-coupled but logically tightly-coupled due to the rules embodied in the component operating systems, e.g., a permanent master/slave relationship is defined. The other reverse condition, tight physical and loose logical coupling, is also possible.

1.3.2 Computer Networks

A computer network can be characterized as a physically loosely-coupled, multiple-computer system in which the interconnection paths have been extended by the inclusion of data communications links. Fundamentally there are no differences between the basic characteristics of computer network systems and other loosely-coupled systems other than the data transfer rates normally provided. The transfer of data between two nodes in the network still requires the active cooperation of both parties involved, but there is no inherently required cooperation between the operation of the processors other than that which they wish to provide.

1.3.3 Distributed Systems

Although there is a large amount of confusion, and often controversy, over exactly what is a "distributed system," it is generally accepted that a distributed system is a multiple computer network designed with some unity of purpose in mind. The processors, databases, terminals, operating systems, and other hardware and software components included in the system have been interconnected for the accomplishment of an identifiable, common goal. That goal may be the supplying of general-purpose computing support, a collection of integrated applications such as corporate management, or embedded computer support such as a real-time process control system.

This research program is concerned with a very specific subclass of all of the systems currently being designated "distributed." The environment of interest here has been given the title "Fully Distributed Processing System" or FDPS. Section 2 discusses the general characteristics of FDPS's.

SECTION 2

INTRODUCTION TO FULLY DISTRIBUTED PROCESSING SYSTEMS

2.1 MOTIVATION OF THE FDPS CONCEPT

A large number of claims have been made as to the benefits that will be achieved with distributed processing systems. As pointed out above, this list is very similar to the lists of "benefits to be achieved" with several earlier computer technologies. However, each of those earlier solutions failed to deliver its promises for various reasons. It was an examination of the "weaknesses" in the earlier concepts and the development of a set of principles to overcome these obstacles that led to the concept of "Fully Distributed Processing Systems" or as it is commonly referred to "FDPS."

The principle of parallel (i.e., simultaneous and/or concurrent) operation of a multiplicity of resources continues to be perhaps the most important goal. The unique feature of FDPS's is the means or environment in which this is attempted. A distributed system should exhibit a continual increase in performance as additional processing components are added. The users should observe shorter response times as well as an increase in total system throughput. In addition, the utilization of system resources should be higher as a result of the system's ability to perform automatic load balancing servicing a large quantity and variety of user work requests. A distributed system should also permit the sharing of data between cooperating users and the making available of specialized resources found only on certain processors. In general, a distributed system should provide more facilities and a wider variety of services than those that can be offered by any system composed of a single processor [Hopp79]. Another important and highly desirable feature of such a system is extensibility. Extensibility might be realized in several different ways. The system might support modular and incremental growth permitting flexibility in its configuration, or it might support expansion in capacity, adding new functions, or both. Finally, it might provide for incremental replacement and/or upgrading of system components, either hardware or software. The executive control of the system is obviously the key to attaining these goals, and it is in the area of executive control that some of the most significant deficiencies of earlier systems have been found.

The major weaknesses in the executive control of earlier forms of parallel systems appear to result from an excessive degree of centralization of control functions reflected in centralized decision making or centralized maintenance of system status information or both of these. The net effect of these aspects of control was to produce a rather tightly-coupled environment in which resources often were idle waiting for work assignments and the failure of one major component often resulted in catastrophic and total system failure. The solution to this problem is to force a condition of very loose coupling on both the logical/control decision making process as well as the physical linkages of components. This property of "universal" loose coupling results in an environment in which the various components are required to operate in an autonomous manner.

If a single design principle must be identified as the most important or central theme of FDPS design, it is component autonomy or "cooperative autonomy" as described below. All of the other features of the definition of Fully Distributed Processing Systems given below have resulted from determining what is required to support and utilize the autonomous operation of the very loosely-coupled physical and logical resources.

2.2 THE DEFINITION OF AN FDPS

Fully Distributed Processing Systems (FDPS) were first defined by Enslow in 1976 [Ensl78] although the designation "fully" was not added until 1978 when it became necessary to clearly distinguish this class of distributed processing from the many others being presented. An FDPS is distinguished by the following characteristics:

1. Multiplicity of resources: an FDPS is composed of a multiplicity of general-purpose resources (e.g., hardware and software processors that can be freely assigned on a short-term basis to various system tasks as required; shared data bases, etc.).
2. Component interconnection: the active components in the FDPS are physically interconnected by a communications network(s) that utilizes two-party, cooperative protocols to control the physical transfer of data (i.e., loose physical coupling).
3. Unity of control: the executive control of an FDPS must define and support a unified set of policies (i.e., rules) governing the operation and utilization or control of all physical and logical resources.

4. System transparency: users must be able to request services by generic names not being aware of their physical location or even the fact that there may be multiple copies of the resources present. (System transparency is designed to aid rather than inhibit and, therefore, can be overridden. A user who is concerned about the performance of a particular application can provide system specific information in order to aid in the formulation of management control decisions.)

5. Component autonomy: both the logical and physical components of an FDPS should interact in a manner described as "cooperative autonomy" [Clar80, Ensl78]. This means that the components operate in an autonomous fashion requiring cooperation among processes for the exchange of information as well as for the provision of services. In a cooperatively autonomous control environment, the components are afforded the ability to refuse requests for service, whether they be execution of a process or the use of a file. This could result in anarchy except for the fact that all components adhere to a common set of system utilization and management policies expressed by the philosophy of the executive control.

2.2.1 Discussion of the Definitional Criteria

In order for a system to qualify as being fully distributed it must possess all five of the criteria presented in this definition.

2.2.1.1 Multiple Resources and Their Utilization

The requirement for resource multiplicity concerns the assignable resources that a system provides. Therefore, the type of resources requiring replication depends on the purpose of a system. For example, a distributed system designed to perform real-time computing for air traffic control requires a multiplicity of special-purpose air traffic control processors and display terminals. It is not required that replicated resources be exactly homogenous, however, they must be capable of providing the same services.

In addition to this multiplicity, it is also required that the system resources be dynamically reconfigurable to respond to a component failure(s). This reconfiguration must occur within a "short" period of time so as to maintain the functional capabilities of the overall system without affecting the operation of components not directly involved. Under normal operation the system must be able to dynamically assign its tasks to components distributed throughout the system.

The extent to which resources are replicated can vary from those systems where none are replicated (not a fully distributed system) to systems where all assignable resources are replicated. In addition, the number of copies of a particular resource can vary depending on the system and type of resource. In general, the greater the degree of replication, particularly of resources in high demand, the greater the potential for attaining benefits such as increased performance (response time and throughput), availability, reliability, and flexibility [Ensl78].

2.2.1.2 Component Interconnection and Communication

The extent of physical distribution of resources in distributed systems can vary from the length of connection between components on a single integrated chip to the distance between two computers connected through an international network. In addition, interconnection organizations can vary from a single bus to a complex mesh network. Since a component in a distributed system communicates with other components through its own logical process, all physical and logical resources can be thought of as processes, and interactions between resources can be referred to as interprocess communication [Davi79]. For example, an application program interacting with processors and data files is accomplished through communication between logical processes.

Both the physical and logical coupling of the system components are characterized as "extremely loose." "Gated" or "master-slave" control of physical transfer is not allowed. Communication, i.e., the physical transfer of messages, is accomplished by the active cooperation of both the sender and addressees. The primary requirement of the intercommunication subsystem is that it support a two-party cooperative protocol. This is essential to enable the system's resources to exist in cooperative autonomy at the physical level.

The advantages of using a message-based (loosely-coupled) communication system with a two-party cooperative protocol include reliability, availability, and extensibility. The disadvantage is the additional overhead of message processing incurred to support this method of communication. There are a variety of interconnection organizations and communication techniques that can be used to support a message-based system with a two-party cooperative protocol.

2.2.1.3 Unity of Control

In a fully distributed data processing system, individual processors will each have their own local operating systems, which may or may not be unique, that control local resources. As a result, control is distributed throughout the system to components that operate autonomously of one another. However, to gain the benefits of distributed processing it is required that the autonomous components of the system cooperate with each other to achieve the overall objectives of the system. To insure this, the concept of a high-level operating system was created to integrate and unify, at least conceptually, the decentralized control of the system.

A high-level operating system is essential to successfully implementing a distributed processing system. This operating system is not a centralized block of code with strong hierarchical control over the system, but rather it is a well-defined set of policies governing the integrated operation of the system as a whole. To insure reliable and flexible operation of the system, these policies should be implemented with minimal binding to any of the system's components [Ensl78].

What policies are required and how they should be implemented depends greatly on the system. For example, if it is a general-purpose system supporting interactive users, then a command interpreter and a user control language will be required to make the system's components compatible and transparent to the user.

2.2.1.4 Transparency of System Control

The high-level operating system also provides the user with his interface to the distributed system. As a result, the user is accessing the system as a whole rather than just a host computer in the network.

In order to increase the effectiveness of the distributed system, the actual system is made transparent, and the user is presented with a virtual machine and a simplified command language to access it. The user uses this language to request services by name and does not have to specify the specific server to be used. Clearly, the same request might be assigned a different server depending on the state of the total system when the request is made. However, to make the system truly effective for all users, knowledgeable individuals must be able to interact with the system more intimately, requesting specific servers or developing service routines to increase the efficiency or effectiveness of the system [Ensl78].

2.2.1.5 Cooperative Autonomy

Cooperative autonomy has already been described at the physical interconnection level. It is also required that all resources be autonomous at the logical control level. That is, a resource must have full control of itself in determining which requests it will service and what future operations it will perform. However, a resource must also cooperate with other resources by operating according to the policies of the high-level operating system. Cooperative autonomy is an essential prerequisite for systems to have fault tolerance and high degrees of extensibility [Ensl78]. It is perhaps the most important as well as the most distinguishing characteristic of a fully distributed processing system.

2.2.2 Effects on System Organization

Although the detailed design of the hardware and software required to implement an FDPS is still in progress, it has been possible for some time to identify certain characteristics that these components must have. One area in which certain criteria already appear reasonably well defined is the nature of the organization of the following system components:

- Hardware
- System control software
- Data bases

It should be noted that a number of definitions and descriptions of distributed systems in general are based on the principle that one or more of these components is physically distributed. (Some such discussions add to this list a fourth component --- "processing or function;" however, considering the distribution of processing independent from the distribution hardware is quite improper. Why distribute the hardware if it will not have some function to perform; similarly, how can the processing be distributed without a corresponding distribution of the hardware? That would be processing on a truly "virtual machine.")

An important characteristic of an FDPS is that, in order to meet the definitional criteria given above while also attempting to provide as many as possible of the benefits listed in Table 1, all of the three components listed above must be physically distributed and the degree of distribution must in each case exceed a reasonably well-defined threshold. A diagram illustrating this requirement is shown in Figure 1. The various organizations of each component identified and positioned along each axis is not meant to be an exhaustive list. These points are listed to better identify the relative location of the three thresholds defining the volume of space occupied by FDPS's. (It might also be noted that it seems quite proper to characterize any system that is not in the "origin cube" as being "distributed" to some degree.)

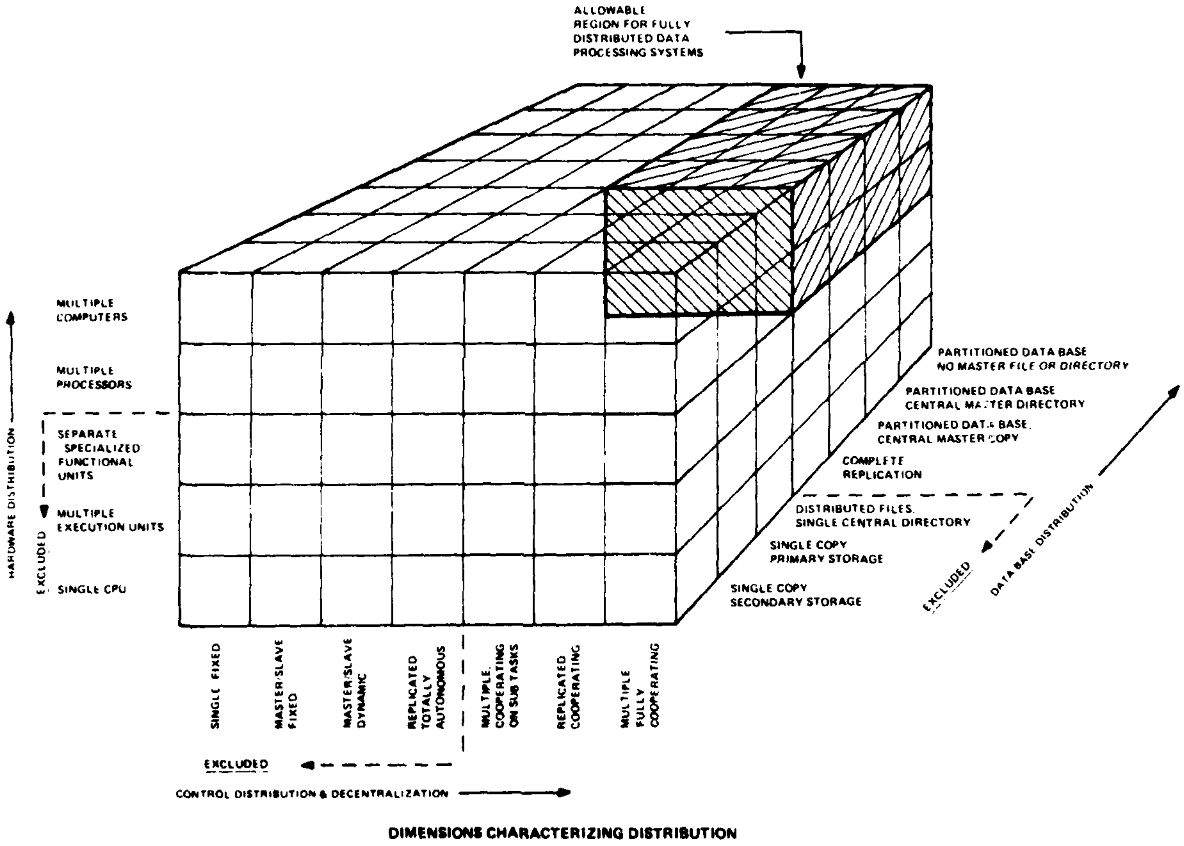


Figure 1. Axes of Distribution

2.3 IMPLICATIONS OF THE FDPS DEFINITION ON CONTROL

2.3.1 General Nature of FDPS Executive Control

Several of the characteristics of an FDPS are found to directly impact the design and implementation of the executive control for such a system. These include system transparency to the user, extremely loose physical and logical coupling, and cooperative autonomy as the basic mode of component interaction. System transparency means that the FDPS appears to a user as a large uniprocessor which has available a variety of services. It must be possible for the user to obtain these services by naming them without specifying any information concerning the details of their physical location. The result is that system control is left with the task of locating all appropriate instances (copies) of a particular resource and choosing the instance to be utilized.

"Cooperative autonomy" is another characteristic of an FDPS heavily impacting its executive control. The "lower-level" control functions of both the logical and physical resource components of an FDPS are designed to operate in a "cooperatively autonomous" fashion. Thus, an executive control must be designed such that any resource is able to refuse a request even though it may have physically accepted the message containing that request. Degeneration into total anarchy is prevented by the establishment of a common set of criteria to be followed by all resources in determining whether a request is accepted and serviced as originally presented, accepted only after bidding/negotiation, or rejected.

Another important FDPS characteristic that definitly affects the design of its executive control is the extremely loose coupling of both physical and logical resources. The components of an FDPS are connected by communication paths of relatively low bandwidth. The direct sharing of primary memory between processors is not acceptable. Even though the logical coupling could still be loose with this physical interconnection mechanism, the presence of a single critical hardware element, the shared memory would create fault-tolerance limitations. All communication takes place over "standard" input/output paths. The actual data rates that can be supported are primarily a function of the distance between processors and the design of their input/output paths. In any event, the transfer rates possible will probably be much less than memory transfer rates. This implies that the sharing of information among components on different processors is greatly curtailed, and system control is forced to work with information that is usually out-of-date and, as a result, inaccurate.

The control of an FDPS requires the action and cooperation of components at all layers of the system. This means that there are elements of FDPS control present in the lowest levels of the hardware as well as software components. This paper is primarily interested in the software components of the FDPS control which are typically referred to as "the executive control."

The executive control is responsible for managing the physical and logical resources of a system. It accepts user requests and obtains and schedules the resources necessary to satisfy a user's needs. As mentioned earlier, these tasks are accomplished so as to unify the distributed components of the system into a whole and provide system transparency to the user.

2.3.2 Why Not Centralized Control?

Why then is a centralized method of control not appropriate? In systems utilizing a centralized *executive control*, all of the control processes share a single coherent and deterministic view of the entire system state. An FDPS, though, contains only loosely-coupled components, and the communication among these components is restricted and subject to variable time delays. This means that one cannot guarantee that all processes will have the same view of the system state [Jens78]. In fact, it is an important characteristic of an FDPS that they will not have a consistent view.

A centralized executive control weakens the fault-tolerance of the overall system due to the existence of a single critical element, the executive control itself. This obstacle, though, is not insurmountable for strategies do exist for providing fault-tolerance in centralized applications. Garcia-Molina [Garc79], for example, has described a scheme for providing fault-tolerance in a distributed data base management system with a centralized control. Approaches of this type typically assume that failures are extremely rare events and that the system can tolerate the dedication of a relatively long interval of time to reconfiguration. These restrictions are usually unacceptable in an FDPS environment where it is important to provide fault-tolerance with a minimum of disruption to the services being supported.

Also, the extremely important issue of overall system performance must be considered. A distributed processing system is expected to utilize a large quantity and a wide variety of resources. If a completely centralized executive control is implemented, there is a high probability that a bottleneck will be created in the node executing the control functions. A distributed and decentralized approach to control attempts to remove this bottleneck by dispersing the control decisions among multiple components on different nodes.

2.3.3 Distributed vs. Decentralized

This paper advocates utilizing an approach for the control of an FDPS that is both distributed and decentralized. There is a clear distinction between the terms "distributed" and "decentralized" as they are used in the context of this project. "*Distributed control*" is characterized by having its *executing components physically located on different nodes*. This means there are *multiple loci of control activity*. In "*decentralized control*," on the other hand, *control decisions are made independently by separate components at different locations*. In other words, there are *multiple loci of control decision making*. Thus, distributed and decentralized control has active components located on different nodes and those components are capable of making independent control decisions.

2.4 AN FDPS APPLICATION --- DATA FLOW PROCESSING

The operating characteristics specified for an FDPS appear to be especially suited to applications composed of cooperating processes that may be executed simultaneously. One class of such applications have been referred to as data flow networks [Denn78, Nels78]. They utilize the independence of the processors combined with the implicit potential for parallel operation of data flow networks to improve performance. In addition to potentially improving performance, the data flow approach often provides a more natural method for expressing a solution to a particular problem. Other systems, including ADAPT [Peeb80], Medusa [Oust80], and TRIX [Ward80], have been designed to service similar types of applications. An application of this type can be expressed either as a command level program [Akin78] or a program in a high level language [Feld79, Macc80]. The execution of individual processes may result from the invocation of files containing either executable code or commands. In such a system, calls to other processes (executable files or command files) can originate from any process, and the nesting of such calls is unlimited.

2.5 PROJECT SCOPE AND ORGANIZATION OF THIS REPORT

Following these two sections of introductory comments, this report discusses the results of an initial study of distributed and decentralized control including, where appropriate, material concerning the results of other projects in the Georgia Tech Research Program on Fully Distributed Processing Systems (FDPS). This initial study of FDPS control has been focused primarily on the qualitative aspects of various forms and implementations of control. The project description is as follows:

> "Define and refine existing models of distributed and decentralized control and develop new models as appropriate to provide a capability of fault tolerance, automatic reconfiguration, and dynamic control."

It is important to note that very few "existing models of distributed control" have been identified and those that have been located are so incompletely defined that this project has proceeded primarily by defining candidate models while attempting to develop a suitable taxonomy of other possible models. Since this project was undertaken fully cognizant that a quantitative study of the models would follow immediately, it is felt that the development of such a taxonomy will help to insure that no significant variations are overlooked.

2.5.1 Discussion of FDPS Models

Along with the development of the various models for distributed and decentralized control, the FDPS team is also developing total system models. These system models provide an essential part of the description of the total environment within which the executive control must operate. Although it is clear at this time that these system models are still evolving, descriptions of their present versions are presented in Section 3.

2.5.2 Issues in Decentralized Control

Although most readers probably have some understanding of the functions of tne executive control in a centralized system, the overall effects of the distributed environment and the set of totally new requirements placed on a decentralized executive control are perhaps not so obvious. The purpose of Section 4 is to discuss the effects of the operating environment and to explicitly identify as many as possible of the new control requirements and limitations as well as variations from centralized control models.

2.5.3 Work Requests

There is a strong relationship between the forms of work requests that the distributed system is expected to process and the capabilities required in the control model. Section 5 focuses on the variations possible in the work requests leaving the discussion of the resulting effects on the operation of the executive control until Section 7.

2.5.4 Characteristics of a Decentralized Control Model

Section 6 of this report presents and discusses those attributes that distinguish various models in the present catalog of decentralized control models. (Note that this is not presented as a complete "taxonomy.") The attributes are characterized in terms of the information that needs to be maintained and the decisions that must be made by an executive control. Also discussed in this section are some of the operational aspects of the models identified thus far.

2.5.5 Control Model Functions

It is during a detailed discussion of the functions performed by an executive control that many of the aspects of decentralized control are best highlighted. In Section 7 discussion of the individual operations are presented and then representative examples of functions such as task graph building are discussed. (A task graph is used to maintain information about the processes being utilized to satisfy a work request. See Paragraph 7.1 for a more complete definition of task graphs.) Experience has shown that many individuals do not fully grasp the significance of distributed and decentralized control until they study examples such as those presented in Section 7.

2.5.6 Example Control Models

A few specific control models that have been developed thus far are presented in Section 8. These include control models advanced by other research teams as well as several developed in the FDPS research program.

2.5.7 Control Model Evaluation

Immediately following this survey of control models the various models will be evaluated. Section 9 presents a preliminary discussion of some of the evaluation criteria to be applied.

SECTION 3

FDPS SYSTEM MODELS

3.1 INTRODUCTION

Models serve extremely important, if not essential, roles in the development of complex systems. This is especially true for systems in which the effects of complexity are further complicated by inconsistencies, ambiguities, and incompleteness in the use of the terms that are employed to describe the structure as well as the operation of the systems involved and the components thereof. Suitable models are valuable, if not essential, tools to support and clarify such discussions. When examining or using any model, it is equally important to recognize that it may have been prepared or developed for a specific purpose (e.g., logical or physical description, simulator design, implementation guide, etc.) and may not be totally suitable for other uses.

3.1.1 Why a "New" Model and "New" Terminology?

Since the concepts of "full distribution" were first conceived over four years ago, members of the FDPS project have been plagued by severe problems in explaining the significance of various aspects of the definition of an FDPS. Most of these problems have been caused by the difficulties in clearly communicating the extremely important differences between "fully" distributed systems and those that are merely "distributed." These problems in understanding appear often to result from the "listener" incorrectly equating certain aspects of FDPS operation with those of a similarly appearing distributed system. Such misunderstandings are not totally unreasonable, for some of the most significant differences are quite subtle. One highly desirable effect anticipated from "new" system models and "new" terminology is to prevent, or at least make less likely, these undesirable associations with existing system concepts.

3.1.2 Approaches to Modelling

There are a number of approaches that may be followed in the development of a system model. The selection of the approach to be taken is based on the intended use of the model and the nature of the system being modelled.

3.1.2.1 Scenario or Flow Chart Models

Certainly one of the most commonly encountered models is the simple flow chart. A flow chart depicts the thread or threads of processing that the system will perform in response to a given set of inputs. A flow chart is probably the best method to illustrate or model the sequence of processing activities involved in a transaction processing or similar type system.

3.1.2.2 Structure Models

Logical and physical structure models are focused more on the organization and modularization of the processing software and hardware than on the actual processing those modules perform. Perhaps the most important use of structure models is in the partitioning of functionality and code for implementation.

3.1.2.3 Interaction Models

Interaction models which focus on the relationships between software and hardware processing entities are becoming quite popular in the area of computer networks; however, they are certainly not limited to just those applications. The basic principle employed in the development of these models is layering with interactions between pairs of peer layers and sets of adjacent layers being specified. The operation and functionality provided by each layer is defined in terms of its protocols and interfaces.

The rules and procedures defining the interactions between peer layers are known as "protocols," whereas "interfaces" define the boundaries and procedures for interaction between adjacent layers. (See Figure 2) (This usage of the term "interface" is consistent with its definition as the boundary between <u>dissimilar entities</u>.) To complete the system description at this level of abstraction, the interfaces are defined in terms of the services provided by a lower layer and the services provided to a higher layer.

It should be noted that in the area of computer networking, the combination of a complete set of protocols and a complete set of interfaces is referred to as a "network architecture."

Preparing a layered model with defined interfaces and protocols is no guarantee that a "clean" layering structure will result. A classic example of this is the ARPANET layers of protocol shown in Figure 3. Although they all make use of the Host-to-IMP protocol, there are many instances in ARPANET in which layers are bypassed completely.

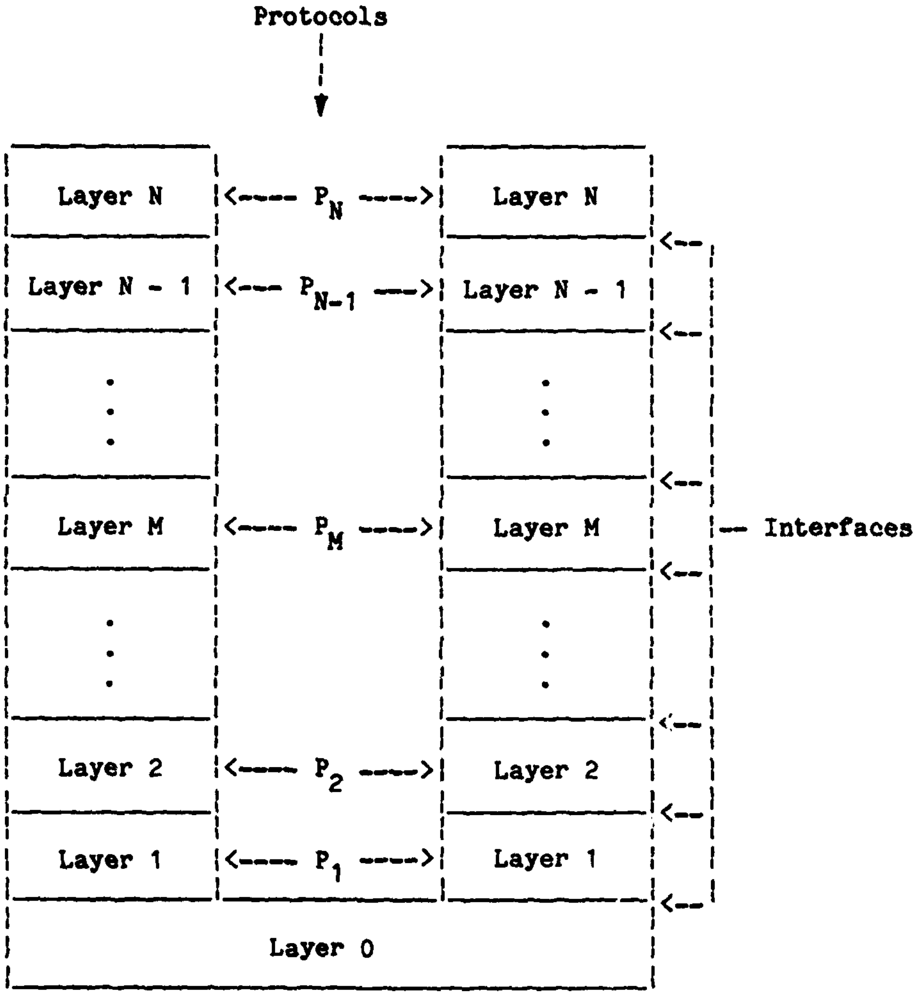


Figure 2. Protocols and Interfaces

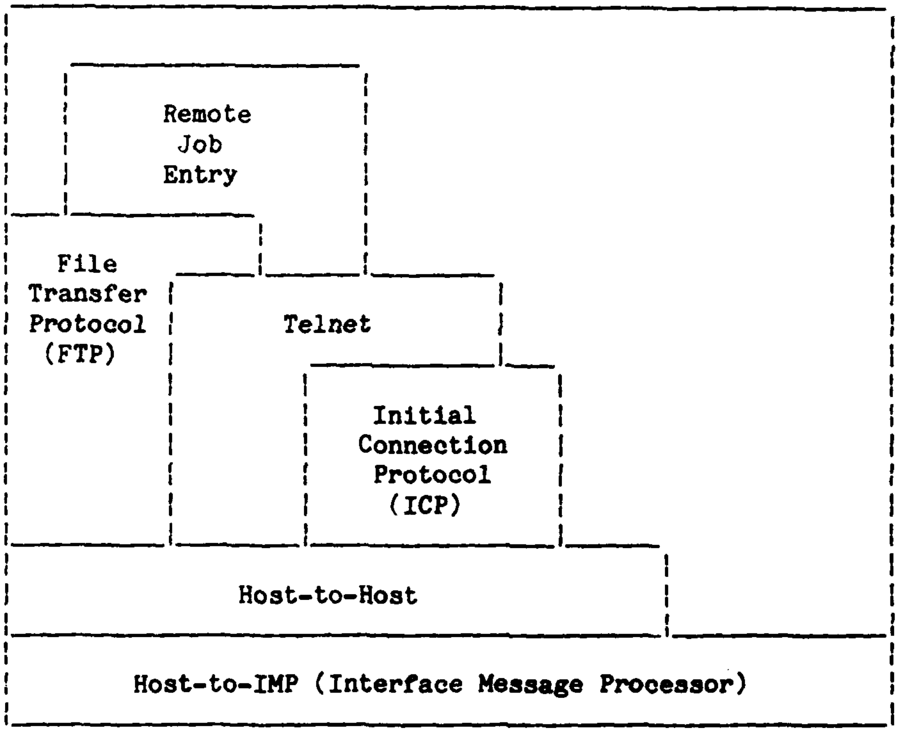


Figure 3. The ARPANET Protocol Layers

3.1.2.4 Performance and Mathematical Models

Obviously, the objective or purpose of this class of **models is to** provide tools to examine, and usually quantify, the performance of **a system.**

3.1.2.5 Summary of Model Types

The various types of models discussed above <u>do not represent different ways to accomplish the same task</u>. Although there is some common **information** found in or derivable from two or more of the various type of models, each is actually focused on quite different aspects of the system description.

- **Physical structure model:** Depicts the manner in which the various hardware and software components are <u>partitioned and packaged</u>.
- **Logical structure model:** Focuses on the <u>functionality</u> provided by the hardware and software components and how they may be logically organized into modules.
- **Scenario or flow chart model:** Depicts the <u>sequence of processing actions</u> taken on the data.
- Interaction model: Focuses on the <u>interactions between processing entities</u> --- services provided to or received from adjacent layer entities and the protocols governing the communication and negotiations that can occur between corresponding peer layers.
- Analytic model: Focuses on the <u>performance of complete systems or subsystems</u>. Often the external performance characteristics of the system being modelled are available.
- Simulation model: Depicts a system or subsystem by <u>modelling as close as possible the operations that it performs</u>. Provides more internal detail than an analytic model.

3.2 <u>OTHER MODELS</u>

Although the work on FDPS models has certainly been strongly influenced by the numerous existing "models" of multiprocessors, multiple computer systems, and computer networks, there has been very little influence from other "distributed system" models since few of these have been developed to the point that they can be closely analyzed. One model that has had a great deal of influence on the development of the FDPS models, at least in guiding the manner in which those models are presented, is the "Reference Model for Open System Interconnection" developed by Sub-Committee 16 of the International Standards Organization Technical Committee 97.

3.2.1 <u>The ISO Reference Model for OSI</u>

The ISO Reference Model, a layered-interaction model, is being prepared by Sub-Committee 16 to establish a framework for the development of standard protocols and interfaces as appropriate for the interconnection of heterogeneous nodes in an "open" computer network and the intercommunication between the processes in these nodes. (This model is almost totally focused on the IPC process, i.e., interprocess communication.) The ISO model is a 7-layer structure as shown in Figure 4.

Although the ISO Reference model has been influential in providing ideas and concepts applicable to a layered model of an FDPS, there are two major factors limiting its direct applicability:

1. The ISO model is almost totally concerned with communication between the nodes of a network. Some references are made to higher level protocols in the applications layer, but these are not a part of the ISO model.
2. Although it is not explicitly stated, there appears to be a general assumption in the ISO model of a degree of coupling that is tighter than that anticipated for an FDPS. (This comment also applies to nearly all of the current network architectures --- even those that include application layer protocols.)

3.2.2 Protocol Hierarchies

As stated above, the ISO Reference Model addresses only a subset of the protocols and interfaces that will be found in a complete distributed system. A more complete picture is shown in Figure 5.

3.3 THE FDPS MODELS

3.3.1 The FDPS Logical Model

The current version of the FDPS logical model is organized into five layers above the "physical interconnection" layer. (Figure 6) The important or significant characteristics of this logical model are:

1. It is also a rudimentary layered-interaction model; however, to be useful, the interaction model must eventually delineate more layers.
2. The operating system has been divided into two parts based on a division of functionality and responsibilities:
   a. The Local Operating System (LOS) is responsible for the detailed control and management of the users and resources at a single node.
   b. The Network Operating System (NOS) is responsible for interactions between this node and all others.

3. The correlation of FDPS layers and ISO layers is the following:

FDPS Layers	ISO Layers
Users and Resources	
Local Operating System	Application
Network Operating System	
Message Handler	Presentation
	Session
	Transport
Message Transporter	Network
	Data Link
	Physical

3.3.2 An FDPS Physical Model

One of the possible physical models for an FDPS operating system is shown in Figure 7. This is a good example of how logical models and physical models may differ in their modularization. In Figure 7, the division between the LOS and NOS layers of the logical model runs horizontal through the MANAGERS in the physical model.

3.3.3 The FDPS Interaction Model

All of the individual layers of the FDPS interaction model have not yet been identified; however, a more detailed list of the protocols that may be loosely related to Figure 5 is given in Figure 8. This list of protocols is especially significant to the FDPS research project since it identifies those specific areas in which work must be done.

```
 ______________                                      ______________
|              |                                    |              |
| Application  |<-----Application Protocols----->| Application  |
|______________|                                    |______________|
|              |                                    |              |
| Presentation |<----Presentation Protocols----->| Presentation |
|______________|                                    |______________|
|              |                                    |              |
|   Session    |<---Session Control Protocols--->|   Session    |
|______________|                                    |______________|
|              |                                    |              |
|  Transport   |<--Transport Control Protocols-->|  Transport   |
|______________|                                    |______________|
|              |                                    |              |
|   Network    |<---Network Control Protocols--->|   Network    |
|______________|                                    |______________|
|              |                                    |              |
|  Data Link   |<------Data Link Protocols------>|  Data Link   |
|______________|                                    |______________|
|              |                                    |              |
|   Physical   |<-------Physical Protocol------->|   Physical   |
|______________|____________________________________|______________|
|                                                                  |
|                      Interconnection Media                       |
|__________________________________________________________________|
```

Figure 4. The ISO Reference Model for OSI

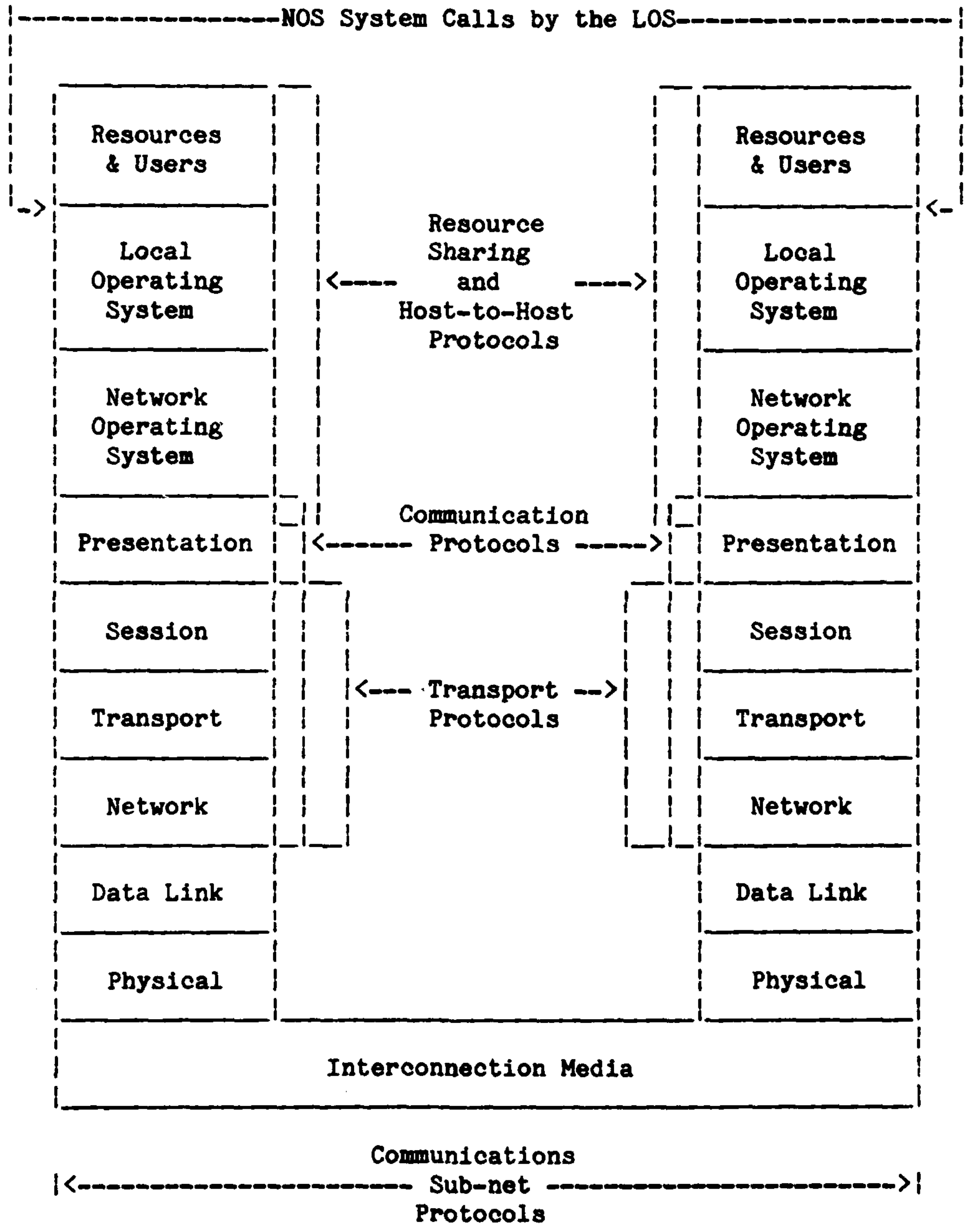


Figure 5. A "Complete" Protocol Hierarchy

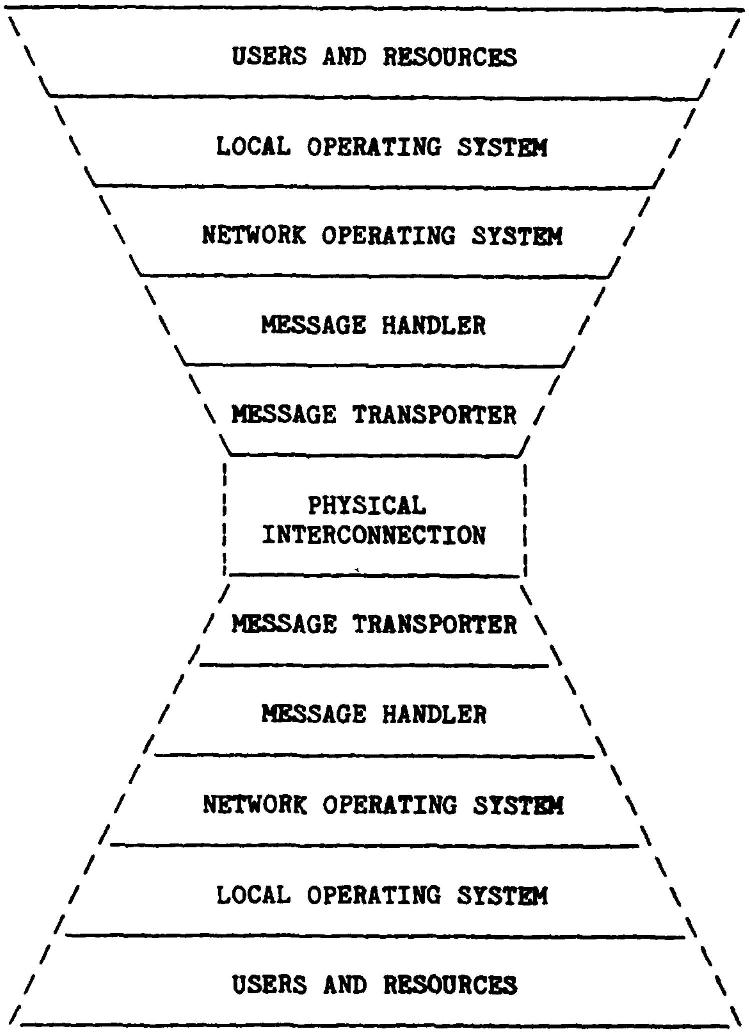


Figure 6. A Logical Model of an FDPS

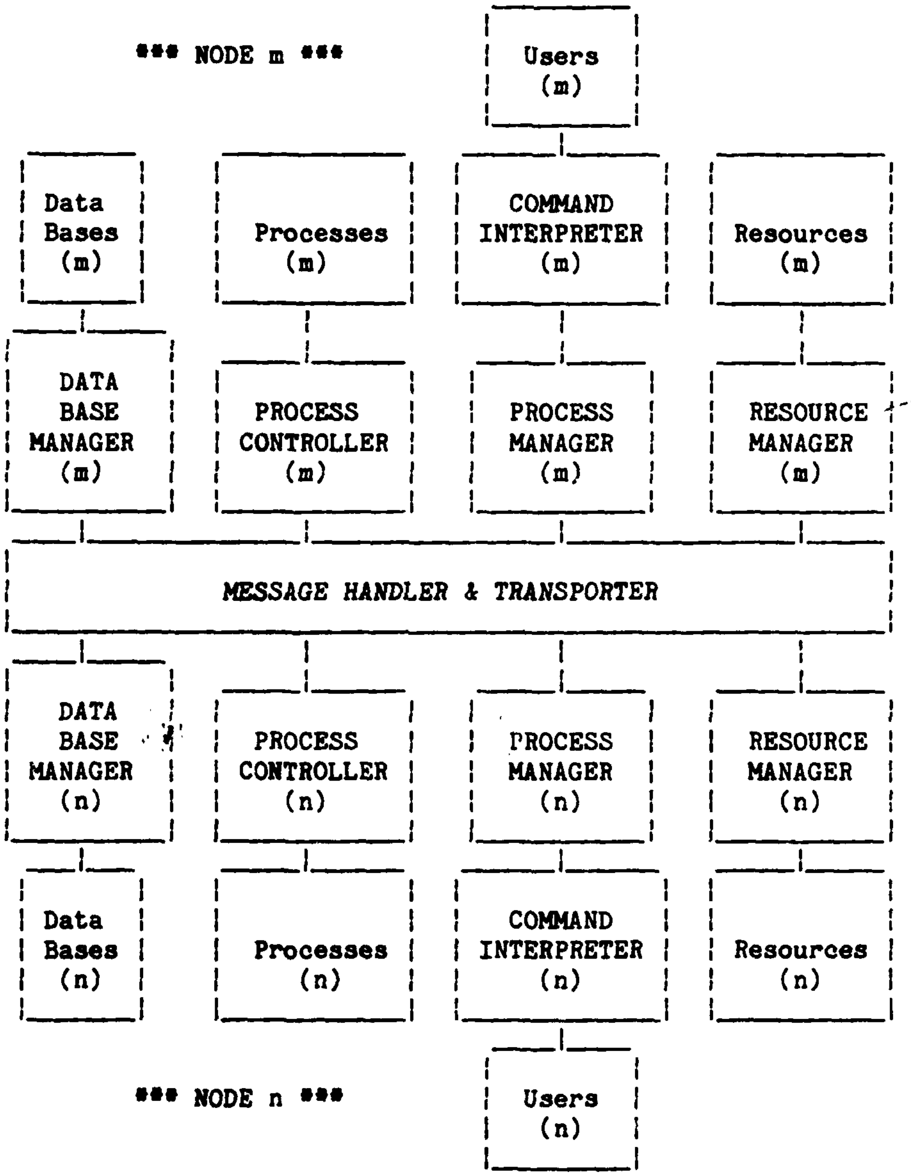


Figure 7. Physical Model of FDPS Control

```
       Computer Network Protocols
                   |
   ________________|_________________
   |                                |
   |                                |
   |                             Resource
Communications                   Sharing
Protocols                        Protocols**
   |                                |
   |-(Processing                    |-(Data Base Control)
   |  Communication)                |  |-File naming
   |  |-Message Formatting          |  |-File access
   |  |-Addressing                  |  |-File transfer
   |                                |  |-Update concurrency
   |-(Message Handling)             |     control
   |  |-Destination                 |
   |  |  resolution                 |-(Access)
   |  |-Connection                  |  |-Virtual terminal
   |  |  establishment              |  |-Access control
   |  |-Message transfer            |  |-User interface
   |                                |     |-Human
   |-(End-to-end)                   |     |-Internal
   |  |-Presentation*               |
   |  |-Session*                    |-(Work Request Processing)
   |                                   |-Resource management
   |-(Transport Subsystem)             |  |-Identification of
   |  |-Transport*                     |  |  resource requirements
   |  |-Network control*               |  |-Resource location
   |  |-Data link*                     |  |-Resource selection
   |  |-Physical*                      |  |-Resource allocation
   |                                   |  |-Resource deallocation
   |-(Communications Subnet)           |-Task management
   |  |-Network control                   |-Execution control
   |  |  |-Routing                        |-Synchronization
   |  |  |-Broadcast                      |-Failure recovery
   |  |-Data link
   |  |-Physical
```

* Classifications (layers) defined by the ISO and CCITT Network Architecture Models

** *A preliminary list for FDPS's*

Figure 8. Classifications of Computer Network Protocols

SECTION 4

ISSUES IN DISTRIBUTED CONTROL

Before examining specific aspects of executive control in an FDPS, a look at some of the various issues of distributed control is appropriate. There are three primary issues that require examination: 1) the effect of the dynamics of FDPS operation on an executive control, 2) the nature of the information an executive control must maintain, and 3) the principles to be utilized in the design of an executive control.

4.1 DYNAMICS

Dynamics is an inherent characteristic of the operation of an FDPS. Dynamics are found in the work load presented to the system, the availability of resources, and the individual work requests submitted. The dynamic nature of each of these provides the FDPS executive control with many unique problems.

4.1.1 Workload Presented to the System

In an FDPS, work requests can be generated either by users or active processes and can originate at any node. Such work requests potentially can require the use of resources on any processor. Thus, the collection of executive control procedures must be able to respond to requests arriving at a variety of locations from a variety of sources. Each request may require system resources located on one or more nodes, not necessarily including the originating node. One of the goals of an FDPS executive control is to respond to these requests in a manner such that the load on the entire system is balanced.

4.1.2 Availability of Resources

Another dynamic aspect of the FDPS environment concerns the availability of resources within the system. As mentioned above, a request for a service to be provided by a system resource may originate at any location in the system. In addition, there may be multiple copies of a resource or possibly multiple resources that provide the same functionality (e.g., there may be functionally equivalent FORTRAN compilers available on several different nodes). Since resources are not immune to failures, the possibility of losing existing resources or gaining both new and old resources exists. Therefore, an FDPS executive control must be able to manage system resources in a dynamic environment in which the availability of a resource is unpredictable.

4.1.3 Individual Work Requests

Finally, the dynamic nature of the individual work requests must be considered. As mentioned above, these work requests define, either directly or indirectly, a set of cooperating processes which are to be invoked. An indirect definition of the work to be done occurs when the work request is itself the name of a command file or contains the name of a command file in addition to names of executable files or directly executable statements. A command file contains a collection of work requests formulated in command language statements (see Figure 10 for a description of the syntax for a suitable command language) that are interpreted and executed when the command file is invoked. The concept of a command file is similar to that of a procedure file which is available on several current systems.

Management of the processes for a work request thus includes the possibility that one or more of the processes are command files requiring command interpretation. The presence of command files will also result in the inclusion of additional information in the task graph or possibly additional task graphs. (See paragraph 7.5 for a discussion of the impact of command files on the task graph.)

An important objective of work request management is to control the set of processes and do so in such a manner that the inherent parallelism present in the operations to be performed is exploited to the maximum. In addition, situations in which one or more of the processes fail must also be handled.

4.2 INFORMATION

All types of executive control systems require information in order to function and perform their mission. The characteristics of the information available to the executive control is one aspect of fully distributed systems that result in the somewhat unique control problems that follow:

1. Because of the nature of the interconnection links and the delays inherent in any communication process, system information on hand is always out of date.
2. Because of the autonomous nature of operation of all components, each processor can make "its own decision" as how to reply to an inquiry; therefore, there is always the possibility that information received is incomplete and/or inaccurate.
3. Because of the inherent time delays experienced in exchanging

information among processes on different nodes, some information held by two processes may conflict during a particular time interval.

4.3 DESIGN PRINCIPLES

Designing the system control functions required for the extremely loosely-coupled environment of an FDPS and implementing those functions to operate in that environment will certainly require the application of some new design principles in addition to those commonly utilized in operating systems for centralized systems. These design principles must address at least the two distinguishing characteristics of FDPS's:

- System information available, and
- Nature of resource control

4.3.1 System Information

The various functions of an FDPS executive control must be designed recognizing that system information is:

- "Expensive" to obtain
- Never fully up-to-date
- Usually incomplete
- Often inaccurate

All of these characteristics of system information result from the fact that the components providing the information are interconnected by relatively narrow bandwidth communication paths (see paragraph 2.3.1) and that those components are operating somewhat autonomously with the possibility that their state may change immediately after a status report has been tansmitted. Further, it is important to note that the mere existence (or disappearance) of a resource is not of interest to a specific component of the FDPS executive control until that component needs that information.

The design principles applying to system information that have been identified thus far include the following:

1. Economy of communication: ask for only the information required.
2. Resiliency: be prepared to recover and continue in the absence of replies.
3. Flexibility: be prepared to recover and continue if the information provided proves to be inaccurate when it is utilized.

4.3.2 Resource Control

Since all of the resources are operating under local control under the policies of cooperative autonomy, all requests for service, or the utilization of any resource such as a file, must be effected through negotiations that culminate in positive acknowledgements by the server. In all instances, the control function requesting a service or a resource must be prepared for refusal.

SECTION 5

CHARACTERIZATION OF FDPS WORK REQUESTS

5.1 THE WORK REQUEST

One of the goals of an FDPS is the ability to provide a hospitable environment for solving problems that allows the user to utilize the natural distribution of data to obtain a solution which may take the form of an algorithm consisting of concurrent processes. The expression of the solution is in terms of a work request that describes a series of cooperating processes, the connectivity of these processes (how the processes communicate), and the data files utilized by these processes. This description involves only logical entities and does not contain any node specific information. A description of one command language capable of expressing requests for work in this fashion can be found in [Akin78] (see Figure 10).

5.2 IMPACT OF THE WORK REQUEST ON THE CONTROL

The nature of allowable work requests (not just the syntax but what can actually be accomplished via the work request) determines to a large extent the functionality of an executive control. Therefore, it is important to examine the characteristics of work requests and further to see how variations in these characteristics impact the strategies utilized by an FDPS executive control.

Five basic characteristics of work requests have been identified:

1. the external visibility of references to resources required by the task,
2. the presence of any interprocess communication (IPC) specifications,
3. the number of concurrent processes,
4. the nature of the connectivity of processes, and
5. the presence of command files.

5.2.1 Visibility of References to Resources

References to the resources required to satisfy a work request may either be visible prior to the execution of a process associated with the work request or embedded in such a manner that some part of the work request must be executed to reveal the reference to a particular resource. A resource is made "visible" either by the explicit statement of the reference in the work request or through a declaration associated with one of the resources referenced in the work request. An example of the latter means of visibility is a file system in which external references made from a particular file are identified and stored in the "header" portion of the file. In this case, the identity of a reference can be obtained by simply accessing the header.

The greatest impact of the visibility characteristic of resource requirements occurs in the construction of task graphs and the distribution of work. The time at which resource requirements are detected and resolved determines when and how parts of the task graph can be constructed. Similarly, some work cannot be distributed until certain details are resolved. For example, consider a case where resource references cannot be resolved until execution time. Assume there exists two processes X and Y where process X has a hidden reference to process Y. An executive control cannot consider Y in the work distribution decision that is made in order to begin execution of X. The significance of this is that certain work distribution decisions may not be "globally optimal" because total information was not available at the time the decision was made.

5.2.2 The Number of Concurrent Processes

A work request can either specify the need to execute only a single process or the execution of multiple processes which may possibly be executed concurrently. Obviously with multiple processes, more resource availability information must be maintained; and there is a corresponding increase in the data to the work distribution and work allocation phases of control. In addition, the complexity of the work distribution decision algorithm increases with more resources needing to be allocated and multiple processes needing scheduling. The complexity of controlling the execution of the work request is also increased with the presence of multiple processes since the control must monitor multiple processes for each work request.

5.2.3 The Presence of Interprocess Communication

The problems described in the previous paragraph are amplified by the presence of communication connections between processes. When interprocess communication is described in a work request, the work distribution decision must consider the requirement for communication links. In addition, a compromise must be made in order to satisfy the conflicting goals of maximizing the inherent parallelism of the processes of the work request and minimizing the cost of communication among these processes. The control activity required during execution is also impacted by the presence of interprocess communication. It must provide the means for passing messages, buffering messages, and providing synchronization to insure that a reader does not underflow and a writer does not overflow the message buffers.

5.2.4 The Nature of Process Connectivity

There are a variety of techniques available for expressing interprocess communication including pipes (see [Ritc78]) and ports (see [Balz71, Have78, Suns77, Zuck77]). There are a number of approaches to realizing these different forms of interprocess communication. The main impact on an executive control, though, is in those components controlling process execution.

5.2.5 The Presence of Command Files

A command file is composed of work requests. Execution of a work request that references a command file results in a new issue dealing with the construction of task graphs. This issue is concerned with whether a new task graph should be constructed to describe the new work request or should these new processes be included in the old task graph. The differences between these two approaches becomes important during work distribution. It is assumed that the work distribution decision will be made only with the information available in the task graph. Thus, with the first approach, only those tasks in the new work request are considered while the second approach provides the ability to take into consideration the assignment of tasks from previous work requests.

5.3 A CLASSIFICATION OF WORK REQUESTS

This examination of the characteristics of FDPS work requests has lead to the identification of five basic attributes which have significant impact on an executive control. In Figure 9, all possible types of work requests are enumerated resulting in 32 different forms of work requests. It should be noted, though, that 16 of these (those with an asterisk beside the task number) contain conflicting characteristics and thus are impossible.

No.	Resource All Visible	References Some Embedded	IPC YES	IPC NO	Resources Distributed on Different Nodes YES	Resources Distributed on Different Nodes NO	Multiple Copies Resources YES	Multiple Copies Resources NO	Some Resources on Node Other Than Home Node YES	Some Resources on Node Other Than Home Node NO
1		X		X		X		X		X
2		X		X		X		X	X	
3*		X		X		X	X			X
4*		X		X		X	X		X	
5*		X		X	X			X		X
6		X		X	X			X	X	
7*		X		X	X		X			X
8		X		X	X		X		X	
9		X	X			X		X		X
10		X	X			X		X	X	
11*		X	X			X	X			X
12*		X	X			X	X		X	
13*		X	X		X			X		X
14		X	X		X			X	X	
15*		X	X		X		X			X
16		X	X		X		X		X	
17	X			X		X		X		X
18	X			X		X		X	X	
19*	X			X		X	X			X
20*	X			X		X	X		X	
21*	X			X	X			X		X
22	X			X	X			X	X	
23*	X			X	X		X			X
24	X			X	X		X		X	
25	X		X			X		X		X
26	X		X			X		X	X	
27*	X		X			X	X			X
28*	X		X			X	X		X	
29*	X		X		X					X
30	X		X		X				X	
31*	X		X		X		X			X
32	X		X		X		X		X	

Figure 9. Classification of Work Requests

SECTION 6

CHARACTERISTICS OF FDPS CONTROL MODELS

6.1 APPROACHES TO IMPLEMENTING FDPS EXECUTIVE CONTROL

There are two basically different approaches available for implementing an operating system for a distributed processing system, the base-level approach and the meta-system approach [Thom78]. The base-level approach does not utilize any existing software and, therefore, requires the development of all new software. This includes software for all local control functions such as memory management and process management. In contrast, the meta-system approach utilizes the "existing" operating systems (called local operating systems (LOS)) from each of the nodes of the system. Each LOS is "interfaced" to the distributed system by a network operating system (NOS) which is designed to provide high level services available on a system-wide basis. The meta-system approach is usually preferred due to the availability of existing software to accomplish local management functions, thus, reducing development costs [Thom78].

Figure 6 depicts a logical model applicable to an FDPS executive control utilizing either approach. The LOS handles the low-level (processor-specific) operations required to directly interface with users and resources. In the meta-system approach, the LOS represents primarily the operating systems presently available for nodes configured in stand-alone environments. The LOS resulting from a base-level approach has similar functionality; however, it represents a new design, and certain features may be modified in order to allow the NOS to provide certain functions normally provided by the LOS. Any "network" operations are performed by the NOS. System unification is realized through the interaction of NOS components, possibly residing on different processors, acting in cooperation with appropriate LOS components. Communication among the components is provided by the message handler which utilizes the message transport services.

6.2 INFORMATION REQUIREMENTS

Two types of information are required by an executive control, information concerning the structure of the set of tasks required to satisfy the work request and information about system resources. This data is maintained in a variety of data structures by a number of different components.

6.2.1 Information Requirements for Work Requests

Each work request identifies a set of cooperating tasks, nodes in a logical network that cooperate in execution to satisfy a request and the connectivity of those nodes. Figure 10 illustrates the notation used in this project to express work requests. An example of a work request using this notation is presented in Figure 11. Work requests as linear textual forms can be easily accepted and manipulated by the computer system; however, task graphs, which are an internal control structure used to describe work requests, must be represented in a manner such that the linkage information is readily available. This can take the form of the explicit linking of node control blocks (Figure 12) or an interconnection matrix (Figure 13).

Information concerning a particular task, i.e., logical node, is maintained in a node control block (Figure 12). Associated with each logical node is an execution file, a series of input files, and a series of output files. The node control block contains information on each of these entities that includes the name of the resource, the locations of possible candidates that might provide the desired resource, and the location of the candidate resource chosen to be utilized in the satisfaction of the work request. In addition to this information, the node control block maintains a description of all interprocess communication (IPC) in which the node is a party. This consists of a list of input ports and output ports. (Interprocess communication is a term describing the exchange of messages between cooperating processes of a work request.) Typically, a message is "sent" when it is written to the output port of a process. The message is then available for consumption by any process possessing an input port that is connected to the previously mentioned output port. The message is actually consumed or accepted when the process owning the connected input port executes a READ on that port.

A global view of interprocess communication is provided by the node interconnection matrix (Figure 13). This structure indicates the presence or absence of an IPC link between an output port of one node and an input port of another node. Thus, links are assumed to carry data in only a single direction.

An example of a task graph resulting from the work request in Figure 11 utilizing the direct linking of node control blocks is presented in Figure 14. Figure 15 illustrates the utilization of an interconnection matrix.

```
<work request> ::= [ <logical net> { ; <logical net> } ]

<logical net> ::= <logical node> { <node separator>
                  { <node separator> } <logical node> }

<node separator> ::= , | <pipe connection>

<pipe connection> ::= [ <port> ] '|' [ <logical node number> ]
                      [ .<port> ]

<port> ::= <integer>

<logical node number> ::= <integer> | $ | <label>

<logical node> ::= [ :<label> ] [ <simple node> |
                   <compound node> ] |
                   ( <simple node> | <compound node> )

<simple node> ::= { <i/o redirector> } <command name>
                  { <i/o redirector> | <argument> }

<compound node> ::= { <i/o redirector> } '{' <logical net>
                    { <net separator> <logical net> } '}'
                    { <i/o redirecotr> }

<i/o redirector> ::= <file name> '>' [ <port> ]  |
                     [ <port> ] '>' <file name>  |
                     [ <port> ] '>>' <file name> |
                                '>>' [ <port> ]

<net separator> ::= ;

<command name> ::= <file name>

<label> ::= <identifier>
```

Figure 10. Work Request Syntax
(Taken from [AKIN78])

Work Request:

```
pgm1 ¦ pgm2 1¦a 2¦b :a pgm3 ¦ pgm4 ¦c.1 :b pgm5 ¦ pgm6 ¦.2 :c pgm7
    (0)     (1) (2) (3)    (4)     (5)   (6)     (7)     (8)  (9)
```

(0) Output port 1 of pgm1 is connected to input port 1 of pgm2.
(1) Ouptut port 1 of pgm2 is connected to input port 1 of the logical node labeled "a," pgm3.
(2) Output port 2 of pgm2 is connected to input port 1 of the logical node labeled "b," pgm5.
(3) Label for the logical node containing pgm3 as its execution module.
(4) Output port 1 of pgm3 is connected to input port 1 of pgm4.
(5) Output port 1 of pgm4 is connected to input port 1 of the logical node labeled "c," pgm7.
(6) Label for the logical node containing pgm5 as its execution module.
(7) Output port 1 of pgm5 is connected to input port 1 of pgm6.
(8) Output port 1 of pgm6 is connected to input port 2 of pgm7.
(9) Label for the logical node containing pgm7 as its execution module.

Data Flow Graph of the Work Request:

```
               pgm1
                ¦
                ¦
                ¦
                V
               pgm2
                ¦¦
           _____¦¦_____
           ¦          ¦
           ¦          ¦
           ¦          ¦
           V          V
          pgm3       pgm5
           ¦          ¦
           ¦          ¦
           V          V
          pgm4       pgm6
           ¦          ¦
           ¦____  ____¦
                ¦¦
                ¦¦
                VV
               pgm7
```

Figure 11. Example of a Work Request

EXECUTION FILE

Name:
Locations of candidates available:
Location of candidate chosen:

INPUT FILE 1

Name:
Locations of candidates available:
Location of candidate chosen:

.
.
.

INPUT FILE i

Name:
Locations of candidates available:
Location of candidate chosen:

OUTPUT FILE 1

Name:
Locations of candidates available:
Location of candidate chosen:

.
.
.

OUTPUT FILE j

Name:
Locations of candidates available:
Location of candidate chosen:

IPC

Input Ports:
Output Ports:

Figure 12. Node Control Block

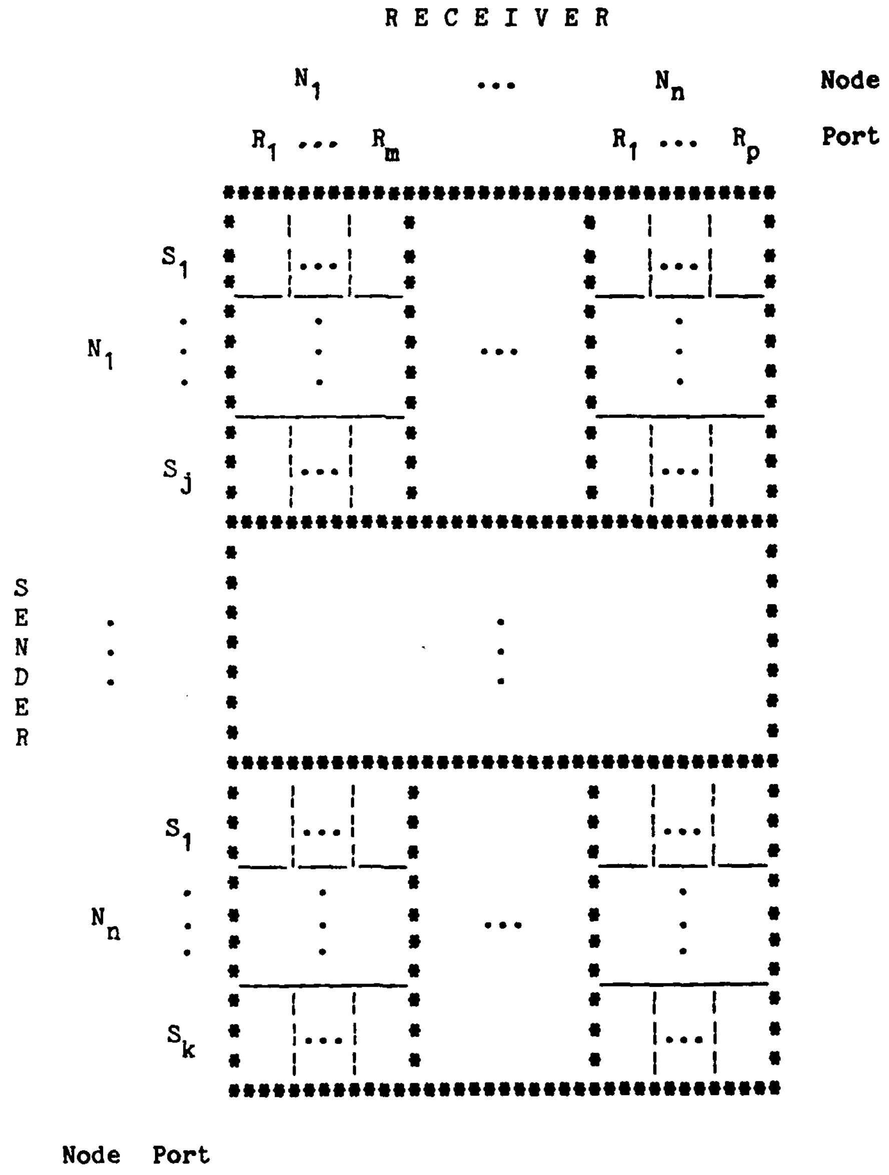


Figure 13. Node Interconnection Matrix

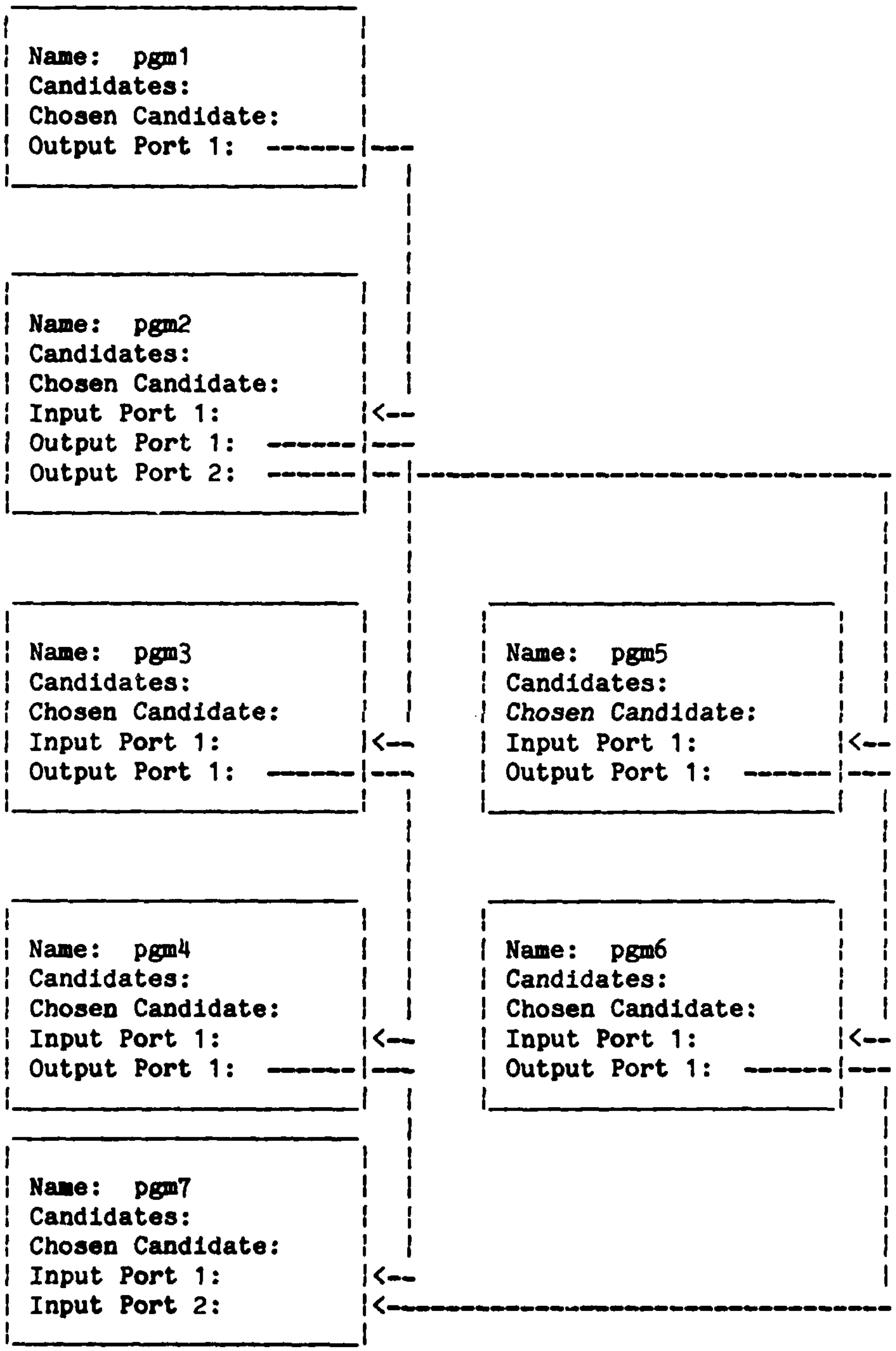


Figure 14. Example of a Task Graph Using Links within the Node Control Blocks

(Based on the Work Request Shown in Figure 11)

RECEIVER

SENDER

		2	3	4	5	6	7		Node
		1	1	1	1	1	1	2	Port
1	1	1							
2	1		1						
	2				1				
3	1			1					
4	1						1		
5	1					1			
6	1							1	
Node	Port								

Figure 15. Example of a Node Interconnection Matrix

(Based on Work Request Shown in Figure 11)

6.2.2 Information Requirements for System Resources

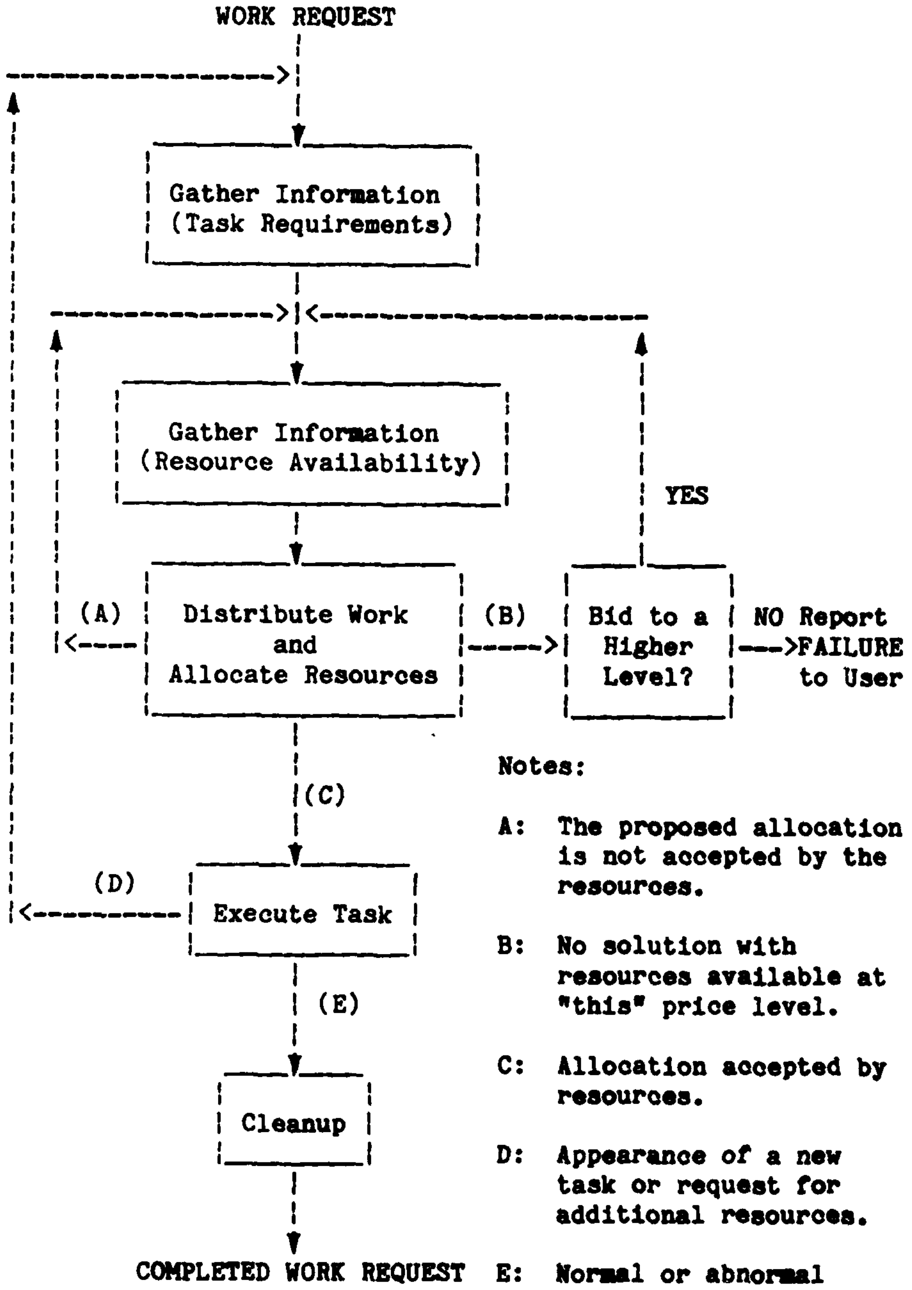
Regardless of how the executive control is realized (i.e., how the components of the executive control are distributed and how the control decisions are decentralized), information concerning all system resources (processors, communication lines, files, and peripheral devices) must be maintained. This information includes at a minimum an indication of the availability of resources (available, reserved, or assigned). Preemptable resources (e.g., processors and communication lines) capable of accommodating more than one user at a time may also have associated with them utilization information designed to guide an executive control in its effort to perform load balancing.

As discussed below, there are a number of techniques that may be employed to gather and/or maintain the system resource information.

6.3 BASIC OPERATIONS OF FDPS CONTROL

The primary task of an executive control is to process work requests that can best be described as logical networks. A node of a logical network specifies an execution file that may either contain object code or commands (work requests), input files, and output files. These files may reside on one or more physical nodes of the system and there may be multiple copies of the same file available. Thus, to process a work request, an FDPS executive control must perform three basic operations: 1) gather information, 2) distribute the work and allocate resources, and 3) initiate and monitor task execution. These operations need not be executed in a purely serial fashion but may take a more complex form with executive control operations executed simultaneously or concurrently with task execution as the need arises.

Examination of the basic operations in further detail (Figure 16) reveals some of the variations possible in the handling of work requests. Two steps exist in information gathering --- 1) collecting information about task requirements for the work request and 2) identifying the resources available for satisfying the request requirements. Information gathering is followed by the task of distributing the work and allocating resources. If this operation is not successful, three alternatives are available. First, more information on resource availability can be gathered in an attempt to formulate a new work distribution. There may have been a change in the status of some resources since the original request for availability information. Second, more information can be gathered as above, but this time the requester will


WORK REQUEST
Gather Information (Task Requirements)
Gather Information (Resource Availability)
Distribute Work and Allocate Resources
(A)
(B)
Bid to a Higher Level?
YES
NO Report FAILURE to User
(C)
Execute Task
(D)
(E)
Cleanup
COMPLETED WORK REQUEST

Notes:

A: The proposed allocation is not accepted by the resources.

B: No solution with resources available at "this" price level.

C: Allocation accepted by resources.

D: Appearance of a new task or request for additional resources.

E: Normal or abnormal termination.

Figure 16. Work Request Processing (Detailed Steps)

indicate a willingness to "pay more" for the resources. This is referred to as bidding to a higher level. Finally, the user can simply be informed that it is impossible to satisfy his work request.

6.3.1 Information Gathering

Upon receiving a work request, the first task of the control is to discover what resources are needed to satisfy the work request (Figure 17) and which resources are available to fill these needs (Figure 18). Each work request includes a description of a series of tasks and the connectivity of those tasks. Associated with each task is a series of files. One is distinguished as the execution file and the rest are input/output files. The executive control must first determine which files are needed. It then must examine each of the execution files to determine the nature of its contents (executable code or commands). Each task will need a processor resource(s), and those tasks containing command files will also require a command interpreter.

An FDPS executive control must also determine which of the system resources are available. For nonpreemptable resources, the status of a resource can be either "available," "reserved," or "assigned." A reservation indicates that a resource may be used in the future and that it should not be given to another user. Typically, there is a time-out associated with a reservation that results in the automatic release of the reservation if an assignment is not made within a specified time interval. The idea here is to free resources that otherwise would have been left unavailable by a lost process. The process may be lost because it failed, its processor failed, or the communication link to the node housing the particular resource may have failed. An assignment, on the other hand, indicates that a resource is dedicated to a user until the user explicitly releases that assignment. Preemptable resources may be accessed by more than one concurrent user and thus can be treated in a different manner. For these resources, the status may be indicated by more continuous values (e.g., the utilization of the resource) rather than the discrete values described above.

6.3.2 Work Distribution and Resource Allocation

The FDPS executive control must determine the work distribution and the allocation of system resources (Figure 19 & 20). This process involves choosing from the available resources those that are to be utilized. This decision

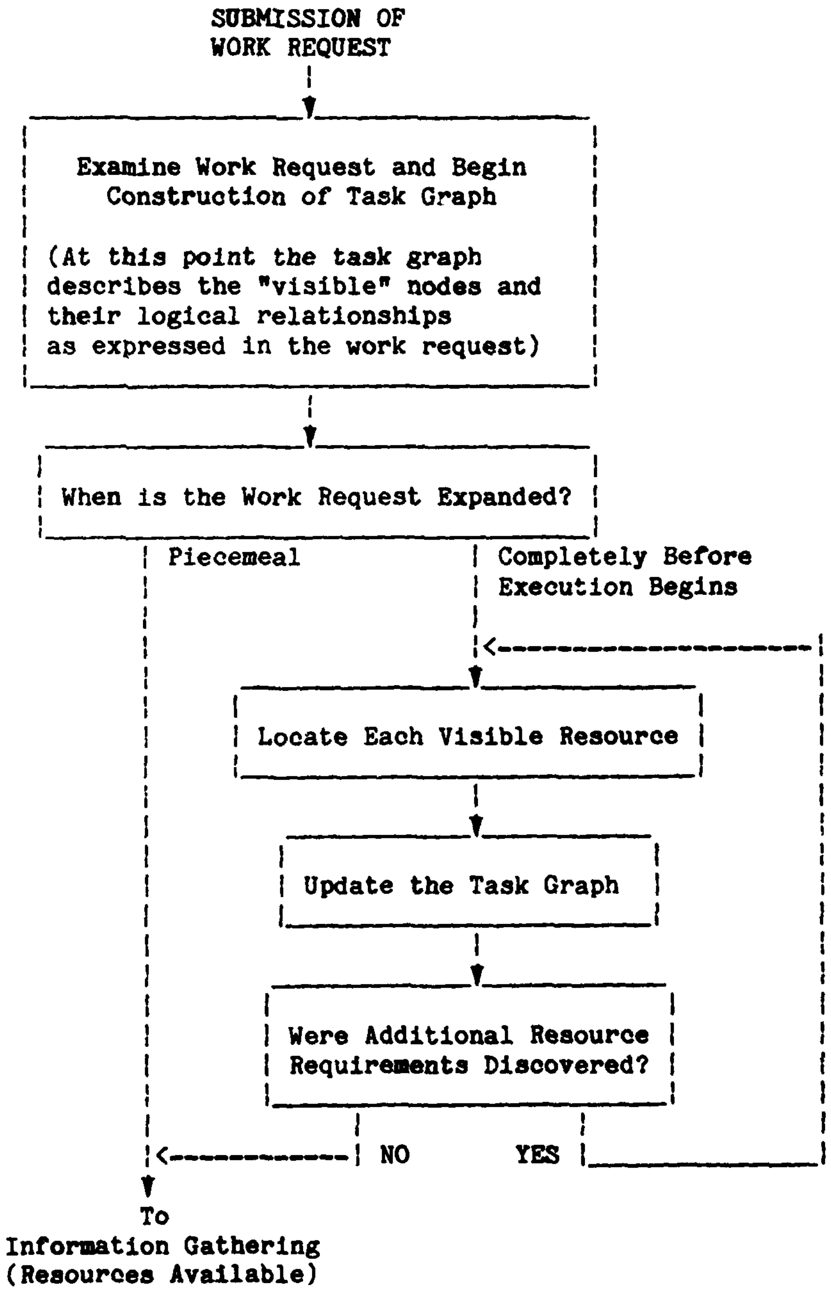
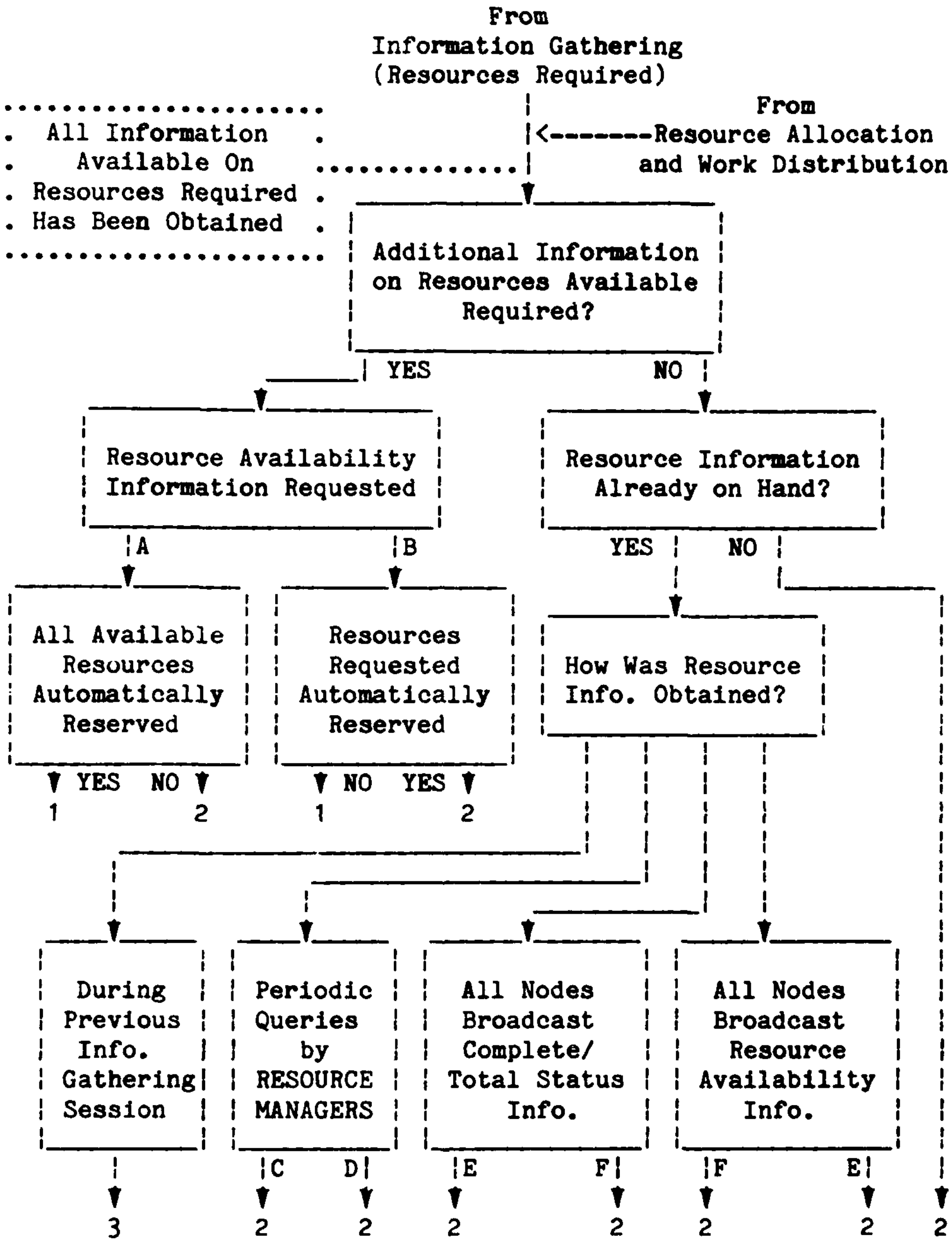

SUBMISSION OF
WORK REQUEST
Examine Work Request and Begin
Construction of Task Graph
(At this point the task graph
describes the "visible" nodes and
their logical relationships
as expressed in the work request)
When is the Work Request Expanded?
Piecemeal
Completely Before
Execution Begins
Locate Each Visible Resource
Update the Task Graph
Were Additional Resource
Requirements Discovered?
NO
YES
To
Information Gathering
(Resources Available)

Figure 17. Information Gathering (Resources Required)



LEGEND AND NOTES

1: Resources Reserved During Information Gathering
2: No Resources Reserved
3: Some Resources May Be Reserved
A: General, for all resources
B: To meet specific task/job requirements
C: Replies cover information on resources available only
D: Replies cover information on the total status
E: Broadcast only significant changes
F: Periodic broadcasts at regular intervals

Figure 18. Information Gathering (Resources Available)

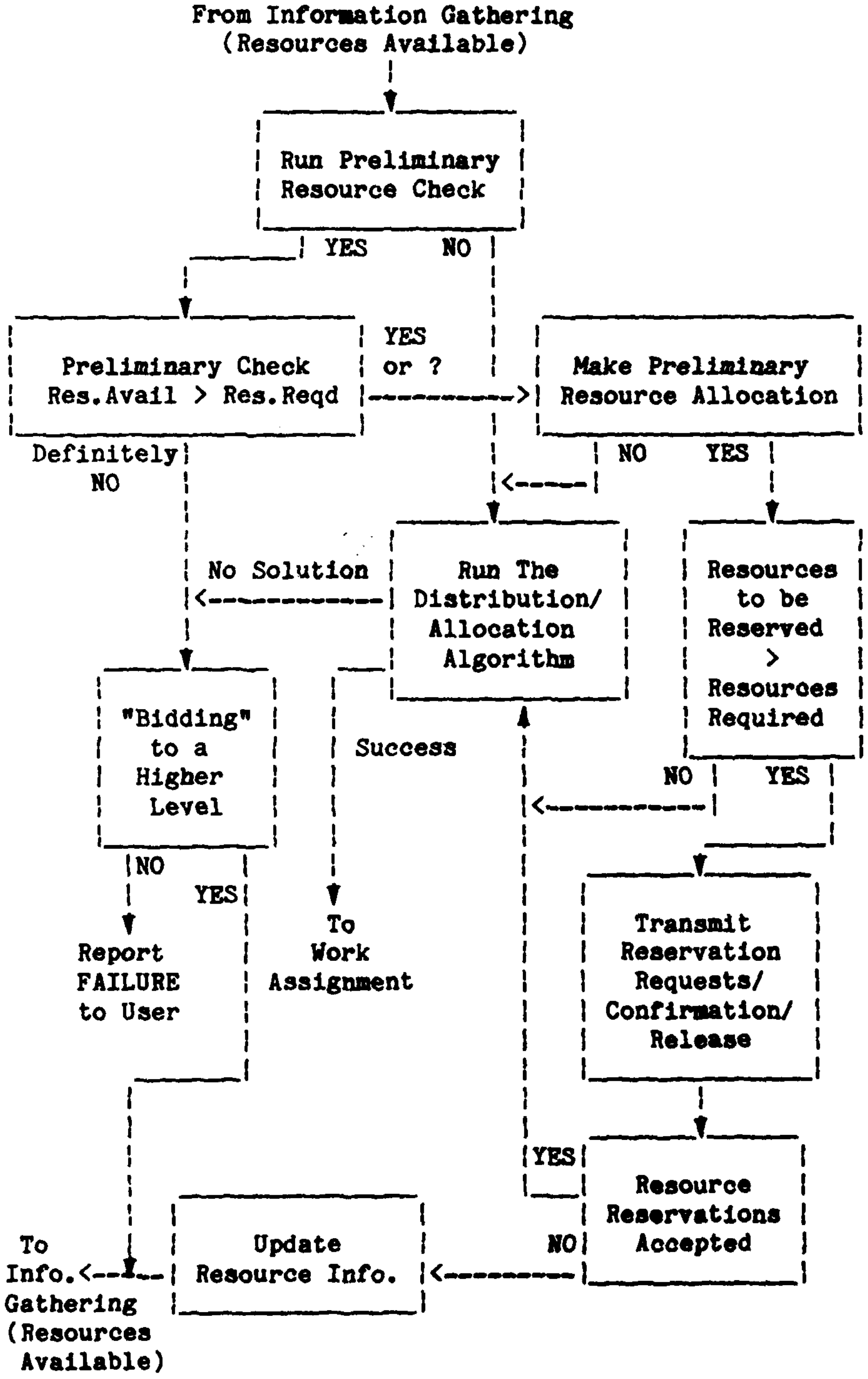

From Information Gathering
(Resources Available)
Run Preliminary Resource Check
YES
NO
Preliminary Check Res.Avail > Res.Reqd
YES or ?
Make Preliminary Resource Allocation
Definitely NO
NO
YES
No Solution
Run The Distribution/ Allocation Algorithm
Resources to be Reserved > Resources Required
"Bidding" to a Higher Level
Success
NO
YES
NO
YES
Transmit Reservation Requests/ Confirmation/ Release
Report FAILURE to User
To Work Assignment
YES
Resource Reservations Accepted
NO
Update Resource Info.
To Info. Gathering (Resources Available)

Figure 19. Resource Allocation and Work Distribution

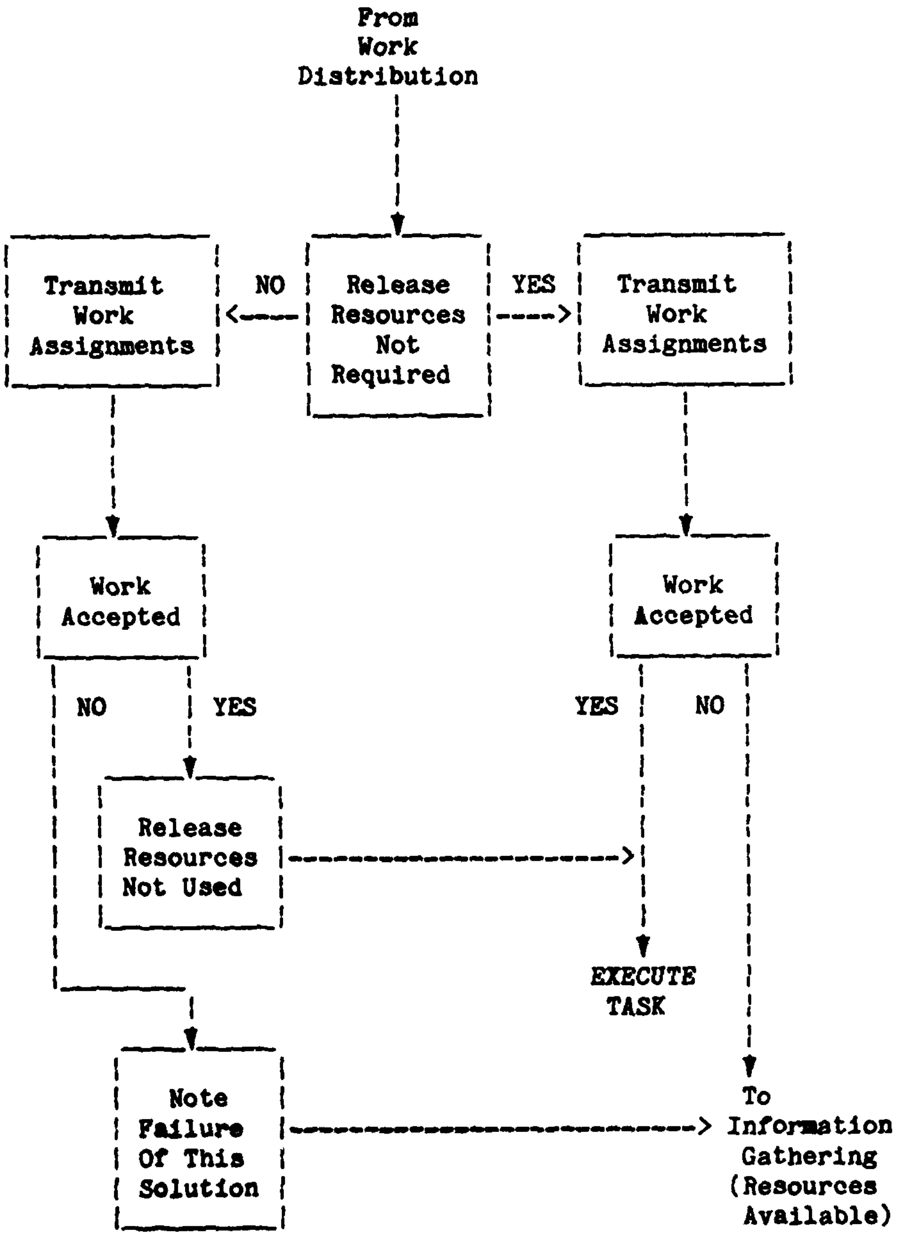

From Work Distribution
Release Resources Not Required
NO
YES
Transmit Work Assignments
Transmit Work Assignments
Work Accepted
Work Accepted
NO
YES
YES
NO
Release Resources Not Used
EXECUTE TASK
Note Failure Of This Solution
To Information Gathering (Resources Available)

Figure 20. Work Assignment

is designed to achieve several goals such as load balancing, maximum throughput, and minimum response time. It can be viewed as an optimization problem similar in many respects to that discussed by Morgan [Morg77].

Once an allocation has been determined, the chosen resources are allocated and the processes comprising the task set are scheduled and initiated. If a process cannot be immediately scheduled, it may be queued and scheduled at a later time. When it is scheduled, a process control block and any other execution-time data structures must be created.

6.3.3 Information Recording

Information is recorded as a result of management actions as well as providing a means to maintain a historical record or audit trail of system activity. The information recording resulting from management actions maintains the system state and provides information for decision making. The historical information is useful in monitoring system security. It provides a means to examine past activity on a system in order to determine if a breach of security occurred or how a particular problem or breach of security may have occurred.

Management information is maintained in various structures, including the task graph. The task graph is used to maintain information about the structure of an individual work request, and, thus, its contents change as progress on the work request proceeds. A task graph is created when a work request is first discovered, and information is then constantly entered into the structure as work progresses through information gathering to work distribution and resource allocation and on to task execution. The task graph remains active until completion of the work request.

Much of the information contained in the task graph is applicable to historical records. In fact, the task graph can be used to house historical information as it is gathered during work request processing. Upon completion of the work request, the historical information is extracted and entered into the permanent historical file. Alternatively, the historical file can be created directly skipping the intermediate task graph structure.

6.3.4 Task Execution

Finally, an executive control must monitor the execution of active processes. This includes providing interprocess communication, handling requests from active processes, and supervising process termination. The activities associated with interprocess communication include establishing communication paths, buffering messages, and synchronizing communicating processes. The latter activity is necessary to protect the system from processes that flood the system with messages before another process has time to absorb the messages. Active processes may also make requests to the executive control. These may take the form of additional work requests or requests for additional resources. Work requests may originate from either command files or files containing executable code.

An executive control must also detect the termination of processes. This includes both normal and abnormal termination. After detecting process termination, it must inform processes needing this information that termination has occurred, open files must be closed, and other loose ends must be cleaned up. Finally, when the last process of a work request has terminated, it must inform the originator of the request of the completion of the request.

6.3.5 Fault Recovery

If portions (tasks) of the work request are being performed on different processors, there is inherently a certain degree of fault recovery possible. The problem is in exploiting that capability. The ability to utilize "good" work remaining after the failure of one or more of the processors executing a work request depends on the recovery agent having knowledge of the location of that work and the ability of the recovery agent to reestablish the appropriate linkages to the new locations for the portions of the work that were being executed on the failed processor(s).

SECTION 7

VARIATIONS IN FDPS CONTROL MODELS

There is an extremely large number of features by which variations in distributed control models can be characterized. Of these, only a few basic attributes appear to deserve attention. These include the nature of how and when a task graph is constructed, the maintenance of resource availability information, the allocation of resources, process initiation, and process monitoring. In this section, these issues are examined; but again, since the number of variations possible in each issue are rather large, only those choices considered significant are discussed. Table 2 contains a summary of the problems that have been identified and possible solutions (significant and reasonable solutions) to these problems.

7.1 TASK GRAPH CONSTRUCTION

The task graph is a data structure used to maintain information about the applicable task set. The nodes of a task graph represent the tasks of the task set, and the arcs represent the connectivity or flow of information between tasks. There are basically four issues in task graph construction: 1) who builds a task graph, 2) what is the basic structure of a task graph, 3) where are the copies of a task graph stored, and 4) when is a task graph built.

The identity of the component or components constructing the task graph is an issue that presents three basic choices. First, a central node can be responsible for the construction of task graphs for all work requests. Another choice utilizes the control component on the node receiving the work request to construct the task graph. Finally, the job of building the task graph can be distributed among several components. In particular, the nodes involved in executing individual tasks of the work request can be responsible for constructing those parts of the task graph that they are processing.

The general nature of the task graph itself provides two alternatives for the design of an executive control. What is of concern here is not the content of a task graph but rather its basic structure. One alternative is to maintain a task graph in a single structure regardless of how execution is distributed. The other choice is to maintain the task graph as a collection of subgraphs with each subgraph representing a part of the work request. For

Table 2. Variations in Control Models

TASK GRAPH CONSTRUCTION:

Who builds the task graph?

1. A central node specializing in task graph building.
2. The node intially receiving and analyzing the work request.
3. All nodes involved in executing the work request.

What is the nature of the task graph?

1. A single complete structure.
2. Multiple structures each consisting of a subgraph.
3. Multiple structures each consisting of a subgraph with one copy of the complete task graph.

Where is the task graph stored?

1. A central node.
2. The node intially receiving and analyzing the work request.
3. A node determined to be in an optimal location.
4. All nodes involved in executing the work request.

When is the task graph built?

1. Completely prior to execution.
2. Piecemeal during execution.

RESOURCE AVAILABILITY INFORMATION:

Who maintains this information?

1. A single central node.
2. Each node maintains information about its own resources.
3. All nodes maintain common information.
4. A designated node for each type of resource.

Where is the information maintained?

1. At a central node.
2. Separate pieces of information concerning a particular resource type may be kept on different nodes.
3. In multiple redundant copies.
4. Information concerning a particular resource type is kept on a specially designated node.

ALLOCATION OF RESOURCES:

How is concurrency control provided?

1. None is provided.
2. Reservations are used prior to a work distribution decision and then allocated by a lock.
3. Allocated by a lock after the work distribution decision.
4. Resources are locked before the work distribution decision is made.

PROCESS INITIATION:

How is responsibility distributed?

1. A central component retains all responsibility.
2. A single component is in charge of a single work request.
3. There is a hierarchy of responsibility.
4. Responsibility is distributed among specialist components.

How is refusal of a request to execute a process by a node handled?

1. After repeated attempts, the request is abandoned.
2. After repeated attempts, a new work distribution is obtained.

PROCESS MONITORING:

What type of interprocess communication is provided?

1. Synchronized communication.
2. Unsynchronized communication.

How are task graphs resulting from additional work requests handled?

1. The new task graph is made part of the old one.
2. The new task graph is kept separate.

PROCESS TERMINATION:

Options selected here are determined by those selected for PROCESS INITIATION.

example, a subgraph can represent that portion of the work request that is to be executed on the particular node at which that subgraph is stored.

Another issue of task graph construction concerns where the various copies of the task graph are stored. If the control maintains a task graph as a unified structure representing the complete set of tasks for a work request, this structure may either be stored on a single node, or redundant copies can be stored on multiple nodes. The single node can either be a central node that is used to store all task graphs, the node at which the original work request arrived (the source node), or a node chosen for its ability to provide this work request with optimal service. If the task graph is divided into several subgraphs, these can be maintained on multiple nodes.

Finally, there is the issue concerning the timing of task graph construction in the sequence of steps that define work request processing. Two choices are available: 1) the task graph can be constructed completely, at least to the maximum extent possible, before execution is begun, or 2) the task graph can be constructed incrementally as execution progresses.

7.2 RESOURCE AVAILABILITY INFORMATION

Another possible source of variability for control models is the maintenance of resource availability information. What is of importance here is "Who maintains this information" and "Where is this information maintained." A particular model need not uniformly apply the same technique for maintaining resource availability information to all resources. Rather, the technique best suited to a particular resource class may be utilized.

The responsibility for maintaining resource availability information can be delegated in a variety of ways. The centralized approach involves assigning a single component this responsibility. In this situation, requests and releases for resources flow through the specialized component which maintains the complete resource availability information in one location.

A variation of this technique maintains complete copies of the resource availability information at several locations [Caba79a,b]. Components at each of these locations are responsible for updating their copy of the resource availability information in order to keep it consistent with the other copies. This requires a protocol to insure that consistency is maintained. For example, two components should not release a file for writing to different users at the same time. To provide this control, messages containing updates for the information tables must be exchanged among the components. In addition, a strategy for synchronizing the release of resources is required. An example oi such a strategy is found in [Caba79a,b] where a baton is passed around the network. The holder of the baton is permitted to release resources.

Another approach exhibiting more decentralization requires dividing the collection of resources into subsets or classes and assigning separate components to each subset. Each component is responsible for maintaining resource availability information on a particular subset. In this case, requests for resources can only be serviced by the control component responsible for that resource. Resources may be named in a manner such that the desired manager is readily identifiable. Alternatively, a search may be required in order to locate the appropriate manager. This search may involve passing the request from component to component until one is found that is capable of performing the desired operation.

Preemptable resources which can be shared by multiple concurrent users (e.g., processors and communication lines) do not necessarily require the maintenance of precise availability information. For these resources, it is reasonable to maintain only approximate availability information because such resources are rarely exhausted. The primary concern in this instance is degraded performance. Therefore, a good estimate of resource utilization is needed.

7.3 ALLOCATING RESOURCES

One of the major problems experienced in the allocation of resources is concurrency control. In a hospitable environment, it is possible to ignore concurrency control. The users are given the responsibility of insuring that access to a shared resource such as a file is handled in a consistent manner. In other environments, for example that presented by an FDPS, this is an important issue. In an FDPS, the problem is even more difficult than in a centralized system due to the loose coupling inherent in the system.

There are basically two approaches to solving the problem of concurrent requests for shared resources. The first utilizes the concept of a reservation. Prior to the allocation of resources (possibly when resource availability information is acquired), a resource may be reserved. The reservation is effective for only a limited period (a period long enough to make a work distribution decision and allocate the resources determined by the decision) and prevents other users from acquiring the resource. The other solution to this problem is to make the work distribution decision without the aid of reservations. If resources cannot be allocated, the executive control will either wait until they can be allocated or attempt a new work distribution.

7.4 PROCESS INITIATION

Several issues arise concerning process initiation. Chief among these is the distribution of responsibility. There are a large number of organizations possible, but only a few are reasonable. The basic organizations utilize either a single manager, a hierarchy of managers, or a collection of autonomous managers. Two approaches result from the single manager concept. In the first organization, a central component is in charge of all work requests and the processes resulting from these work requests. All decisions concerning the fate of processes and work requests are made by this component. A variation on this organization assigns responsibility at the level of work requests. In other words, separate components are assigned to each work request. Each component makes all decisions concerning the fate of a particular work request and its processes.

Management can also be organized in a hierarchical manner. There are a variety of ways hierarchical management can be realized, but we will concentrate on only two, the two-level hierarchy and the n-level hierarchy. The two-level hierarchy has at the top level a component that is responsible for an entire work request. At the lower level are a series of components each responsible for an individual task of the work request. The lower level components take direction from the high level component and provide results to this component. The n-level hierarchy utilizes in its top and bottom levels the components described for the two-level hierarchy. The middle levels are occupied by components that are each responsible for a subgraph of the entire task graph. Therefore, a middle component takes direction from and reports to a higher level component which is in charge of a part of the task graph that includes the subgraph for which the middle component is responsible. The middle component also directs lower level components each of which are responsible for a particular task.

Another organizational approach utilizes a series of autonomous management components. Each component is in charge of some subset of the tasks of a work request. Cooperation of the components is required in order to realize the orderly completion of a work request.

Regardless of the organization, at some point, a request for the assumption of responsibility by a component will be made. Such a request may be reasonably denied for two reasons: 1) the component does not possess enough resources to satisfy the request (e.g., there may not be enough space to place a new process on an input queue), or 2) the component may not be functioning. The question that arises concerns how this denial is handled. One solution is to keep trying the request either until it is accepted or until a certain number of attempts have failed. In this case if the request is never accepted, the work request is abandoned, and the user is notified of the failure. Instead of abandoning the work request, it is possible that a new work distribution decision can be formulated utilizing the additional knowledge concerning the failure of a certain component to accept a previous request.

7.5 PROCESS MONITORING

The task of monitoring process execution presents the FDPS executive control with two major problems, providing interprocess communication and responding to additional work requests and requests for additional resources. With regard to the problem of interprocess communication, there is some question as to the nature of the communication primitives an FDPS executive control should provide. This question arises due to the variety of communication techniques being offered by current languages. There are two basic approaches found in current languages, synchronized communication and unsynchronized communication (buffered messages). Synchronized communication requires that the execution of both the sender and the receiver be interrupted until a message has been successfully transferred. Examples of languages utilizing this form of communication are Hoare's Communicating Sequential Processes [Hoar78] and Brinch Hansen's Distributed Processes [Brin78]. In contrast, buffered messages allow the asynchronous operation of both senders and receivers. Examples of languages using this form of communication are PLITS [Feld79] and STARMOD [Cook80].

The executive control is required to provide communication primitives that are suitable to one of the communication techniques discussed above. If the basic communication system utilizes synchronized communication, both techniques can be easily handled. The problem with this approach is that there is extra overhead incurred when providing the message buffering technique. On the other hand if the basic communication system utilizes unsynchronized communication, there will be great difficulty in realizing a synchronized form of communication.

The task of monitoring processes also involves responding to requests generated by the executing tasks. These may be either requests for additional resources (e.g., an additional file) or new work requests. If the request is a work request, there is a question as to how a new set of tasks is to be associated with the existing set of tasks. The new set could either be included in the existing task graph, or a new task graph could be constructed for these new tasks. The former technique allows the component making the work distribution decision for the new work request to consider the utilization of other resources by the control. The latter technique does not allow such a situation to occur.

7.6 PROCESS TERMINATION

When a process terminates there is always some cleanup work that must be accomplished (e.g., closing files, returning memory space, and deleting records concerning that process from the executive control's work space). In addition, depending on the reason for termination (normal or abnormal), other control components may need to be informed of the termination. In the case of a failure, the task graph will contain the information needed to perform cleanup operations (e.g., the identities of the processes needing information concerning the failure). Both the nature of the cleanup and the identity of the control components that must be informed of the termination are determined from the design decisions resulting from the issues discussed in Section 7.5.

7.7 EXAMPLES

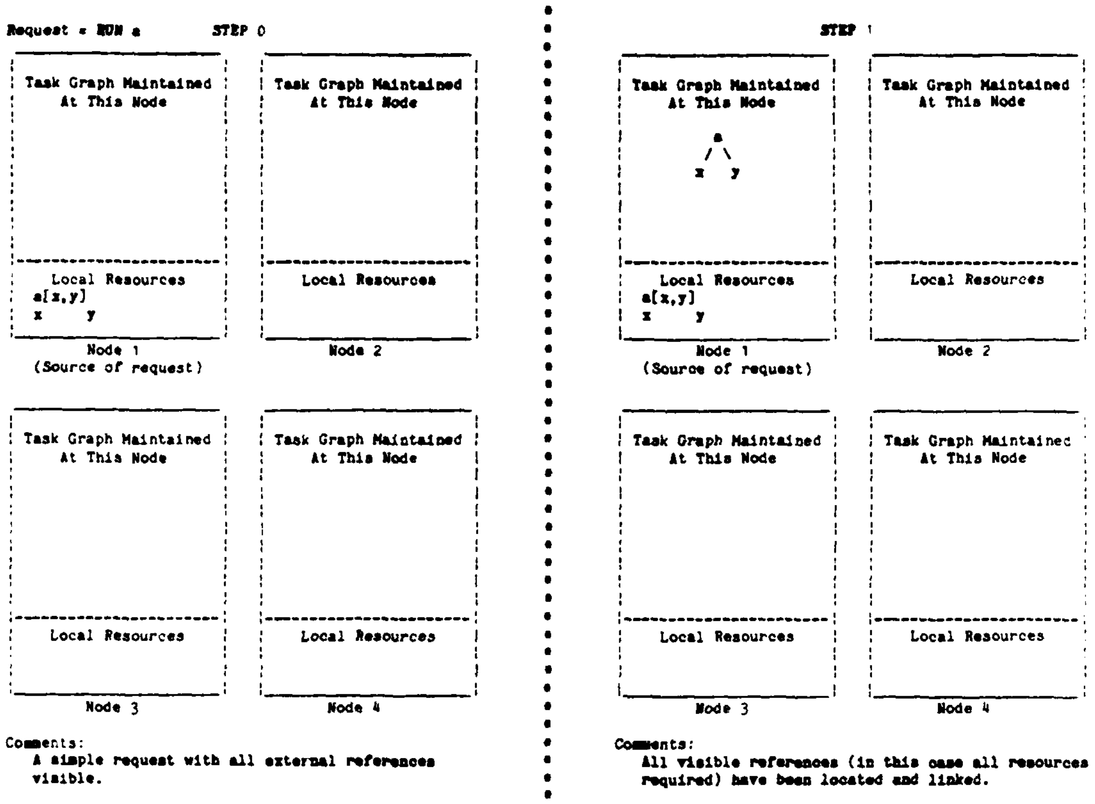
To gain a better appreciation of some of the basic issues of control in an FDPS, it is useful to examine several examples of work request processing on an FDPS. In each example, emphasis is placed on the operations involved in the construction of task graphs. The work distribution decision that is utilized is a simple one that assigns the execution of processes to the same nodes that house the files containing their code. The concern of the first eight examples is the impact of variations in work requests on task graph construction. In these examples, the various parts of the overall task graph describing the complete work request are stored on the nodes utilized by each part. The last three examples, though, examine three different techniques for storing the task graphs. In the examples (Figures 21 to 31) the following symbols are utilized:

[ ]	visible external reference(s)
{ }	embedded external reference(s)
(n)A	responsibility for A delegated from node n
A(n)	responsibility for A delegated to node n
a-->b	IPC from process a to process b
A,B,...	uppercase letters indicate command files
a,b,...	lowercase letters indicate executable files
u,v,w,x,y,z	indicate data files

The first example (Figure 21) consists of a simple request in which all external references made are visible and all files required are present on the node where the original request arrived (referred to as the source node). Since the references are visible, the entire task graph can be completed in one step. The second example (Figure 22) is similar to the first except that there are more references that are chained. Again, since all references are visible, the entire task graph can be completed in one step. This work request can be processed in an alternate manner as shown by the third example (Figure 23) where references are located and linked in a piecemeal fashion. Finally, example 4 (Figure 24) adds a slight variation by introducing an explicit interprocess communication (IPC) definition. In this case, the task graph can still be constructed in one step because all references are visible.

The next series of examples consider the impact of locating resources on nodes other than the source node. In example 5 (Figure 25), all the referenced resources reside on a single node other than the source node with the exception of one resource that has redundant copies on two different nodes. Since the resources are not on the source node, negotiation is required to transfer responsibility for a piece of the task graph. In addition, since there is a resource with two redundant copies, a decision as to which to utilize must be made and a negotiation must occur to transfer responsibility. Example 6 (Figure 26) is similar to example 5 and demonstrates the impact of IPC across nodes.

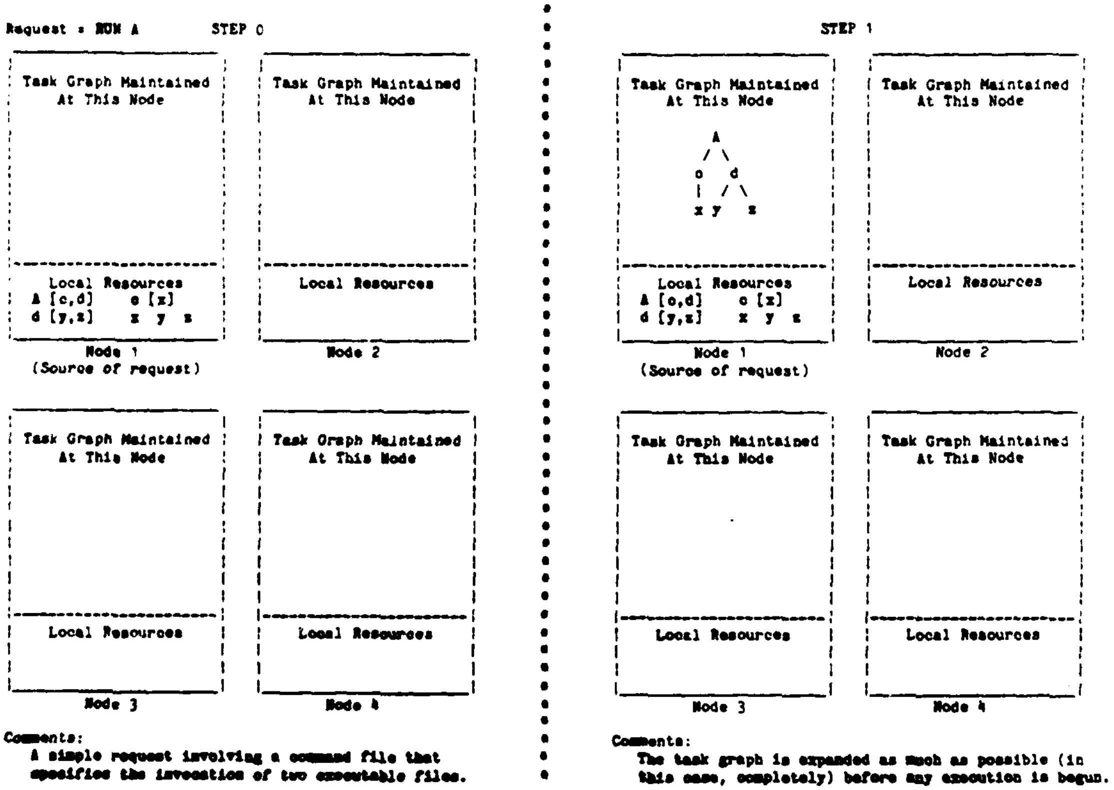
The effect of embedded references is demonstrated in examples 7 and 8. In example 7 (Figure 27), all resources turn out to reside on the source node.


Request = RUN a
STEP 0
Task Graph Maintained At This Node
Local Resources
a[x,y]
x y
Node 1
(Source of request)
Task Graph Maintained At This Node
Local Resources
Node 2
Task Graph Maintained At This Node
Local Resources
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4
STEP 1
Task Graph Maintained At This Node
a
/ \
x y
Local Resources
a[x,y]
x y
Node 1
(Source of request)
Task Graph Maintained At This Node
Local Resources
Node 2
Task Graph Maintained At This Node
Local Resources
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4

Comments:
A simple request with all external references visible.

Comments:
All visible references (in this case all resources required) have been located and linked.

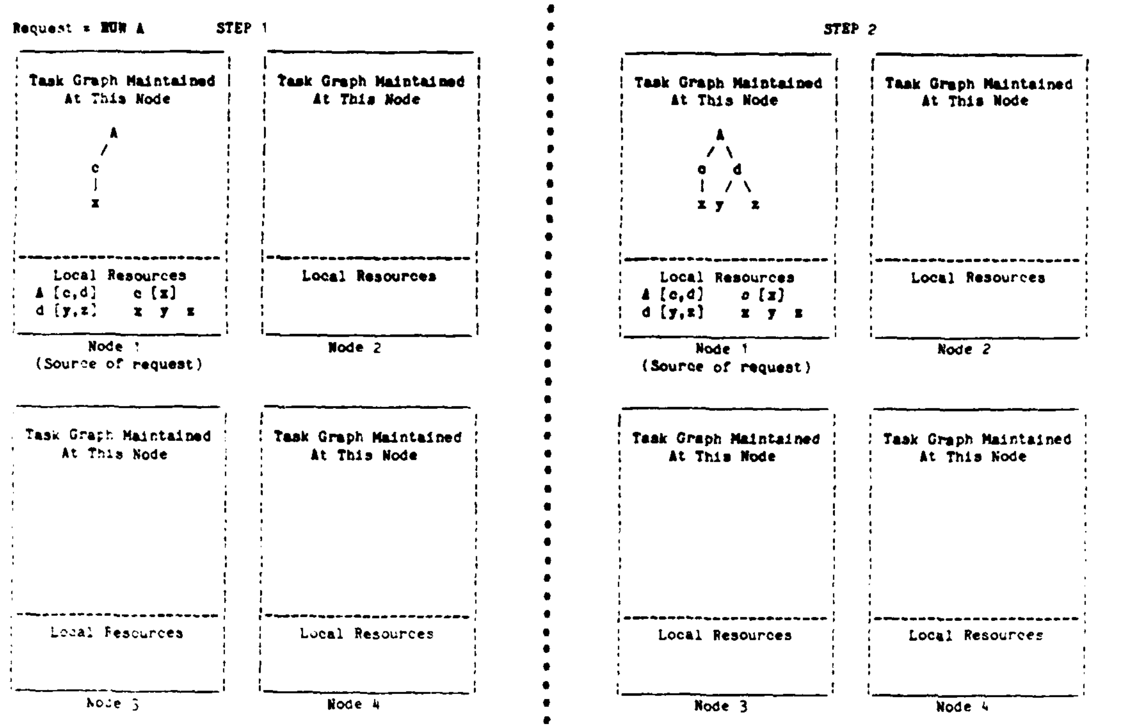
Figure 21. Example 1



Comments:
A simple request involving a command file that specifies the invocation of two executable files.

Comments:
The task graph is expanded as much as possible (in this case, completely) before any execution is begun.

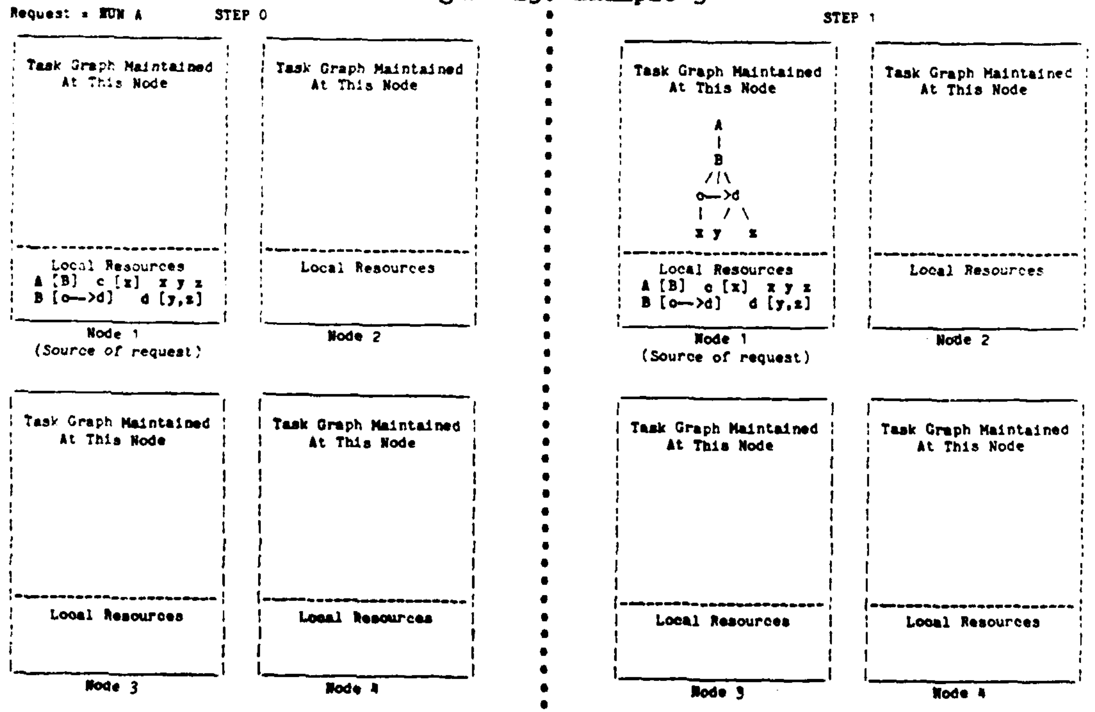
Figure 22. Example 2



Comments:
External references are not located and linked until they are required during execution.

Comments:
All external references have been located and linked.

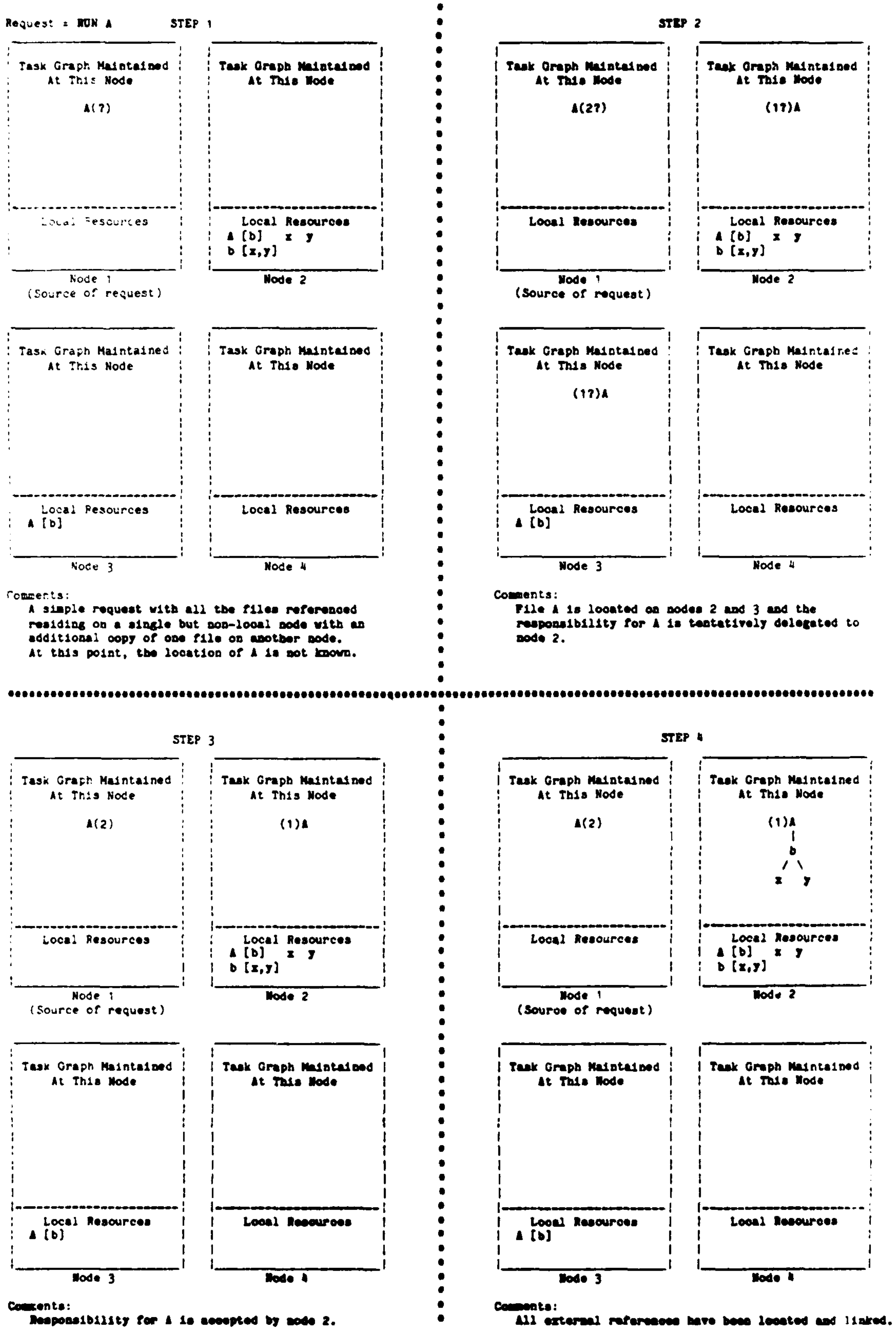
Figure 23. Example 3



Comments:
A somewhat more complex request:
1) All external references visible.
2) Chain of references are present.
3) Contains an explicit IPC definitio..

Comments:
All external references are located and linked, and IPC is established.

Figure 24. Example 4


Request = RUN A
STEP 1
Task Graph Maintained At This Node
A(?)
Local Resources
Node 1 (Source of request)
Task Graph Maintained At This Node
Local Resources
A [b] x y
b [x,y]
Node 2
Task Graph Maintained At This Node
Local Resources
A [b]
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4
STEP 2
Task Graph Maintained At This Node
A(2?)
Local Resources
Node 1 (Source of request)
Task Graph Maintained At This Node
(1?)A
Local Resources
A [b] x y
b [x,y]
Node 2
Task Graph Maintained At This Node
(1?)A
Local Resources
A [b]
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4
STEP 3
Task Graph Maintained At This Node
A(2)
Local Resources
Node 1 (Source of request)
Task Graph Maintained At This Node
(1)A
Local Resources
A [b] x y
b [x,y]
Node 2
Task Graph Maintained At This Node
Local Resources
A [b]
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4
STEP 4
Task Graph Maintained At This Node
A(2)
Local Resources
Node 1 (Source of request)
Task Graph Maintained At This Node
(1)A
b
x y
Local Resources
A [b] x y
b [x,y]
Node 2
Task Graph Maintained At This Node
Local Resources
A [b]
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4

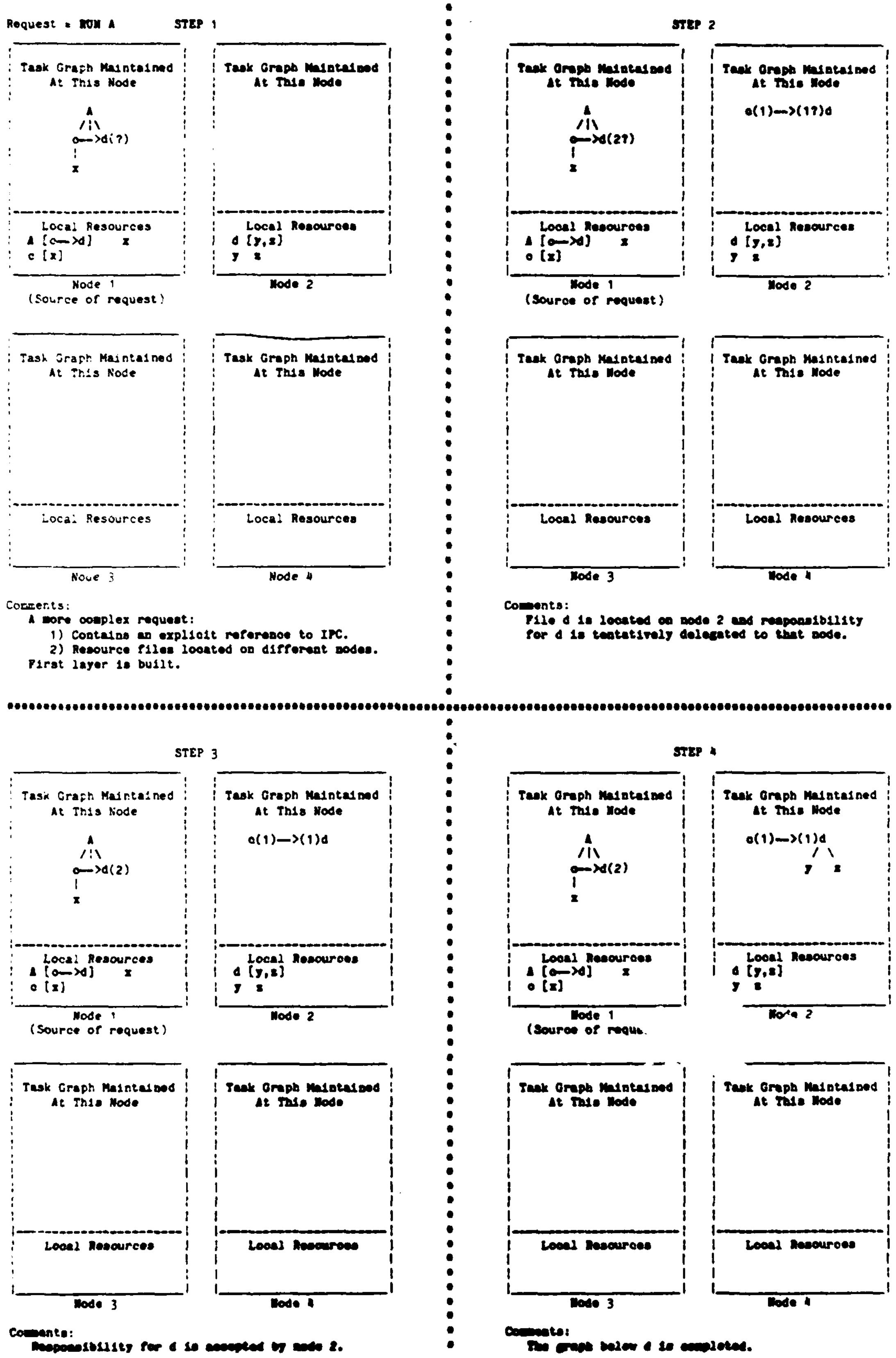
Comments:
A simple request with all the files referenced residing on a single but non-local node with an additional copy of one file on another node. At this point, the location of A is not known.

Comments:
File A is located on nodes 2 and 3 and the responsibility for A is tentatively delegated to node 2.

Comments:
Responsibility for A is accepted by node 2.

Comments:
All external references have been located and linked.

Figure 25. Example 5



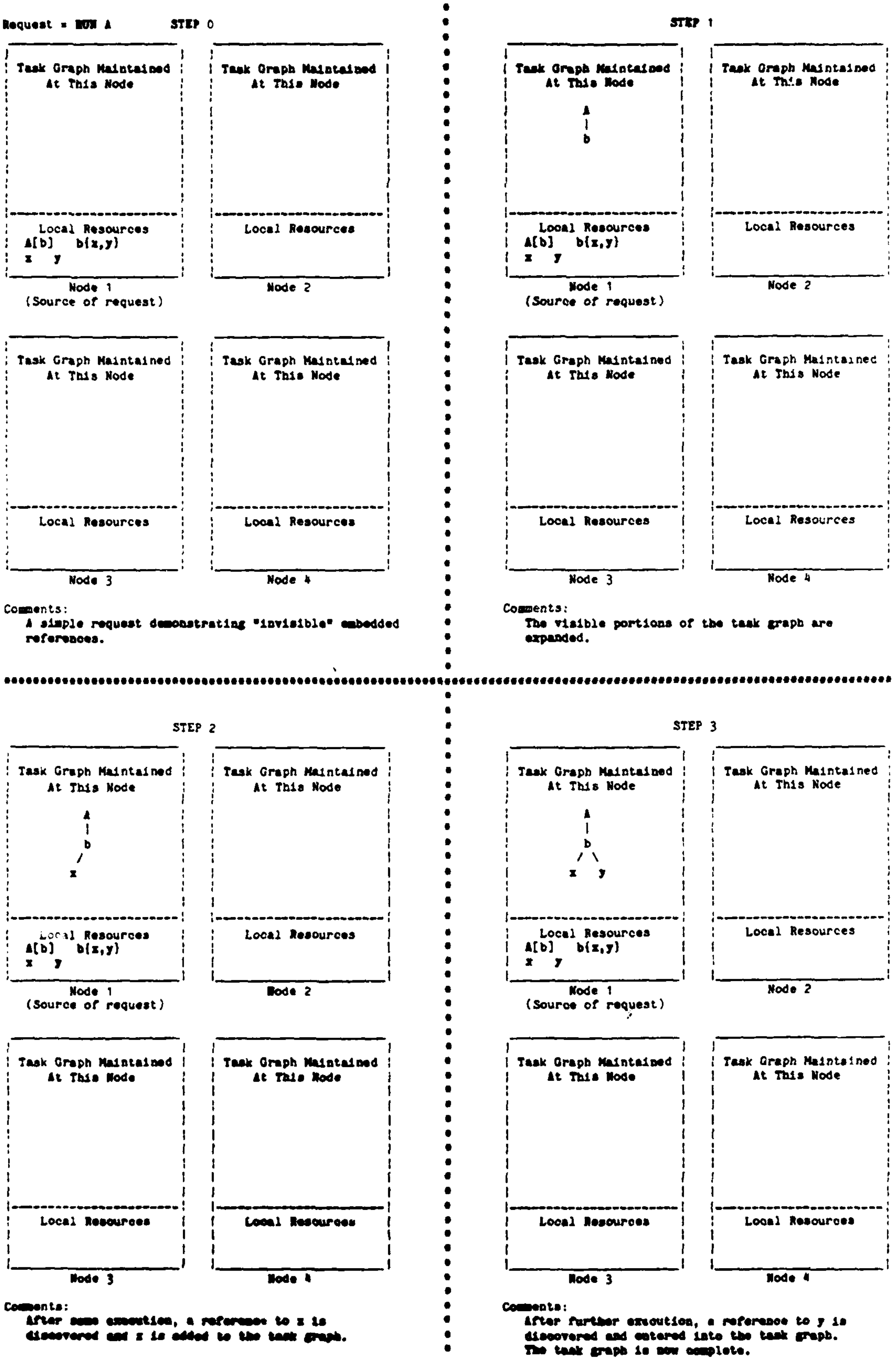
Comments:
A more complex request:
1) Contains an explicit reference to IPC.
2) Resource files located on different nodes.
First layer is built.

Comments:
File d is located on node 2 and responsibility for d is tentatively delegated to that node.

Comments:
Responsibility for d is accepted by node 2.

Comments:
The graph below d is completed.

Figure 26. Example 6


Request = RUN A
STEP 0
Task Graph Maintained At This Node
Local Resources
A[b] b(x,y)
x y
Node 1
(Source of request)
Task Graph Maintained At This Node
Local Resources
Node 2
Task Graph Maintained At This Node
Local Resources
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4
STEP 1
Task Graph Maintained At This Node
A
b
Local Resources
A[b] b(x,y)
x y
Node 1
(Source of request)
Task Graph Maintained At This Node
Local Resources
Node 2
Task Graph Maintained At This Node
Local Resources
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4
STEP 2
Task Graph Maintained At This Node
A
b
x
Local Resources
A[b] b(x,y)
x y
Node 1
(Source of request)
Task Graph Maintained At This Node
Local Resources
Node 2
Task Graph Maintained At This Node
Local Resources
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4
STEP 3
Task Graph Maintained At This Node
A
b
x y
Local Resources
A[b] b(x,y)
x y
Node 1
(Source of request)
Task Graph Maintained At This Node
Local Resources
Node 2
Task Graph Maintained At This Node
Local Resources
Node 3
Task Graph Maintained At This Node
Local Resources
Node 4

STEP 0 Comments:
A simple request demonstrating "invisible" embedded references.

STEP 1 Comments:
The visible portions of the task graph are expanded.

STEP 2 Comments:
After some execution, a reference to x is discovered and x is added to the task graph.

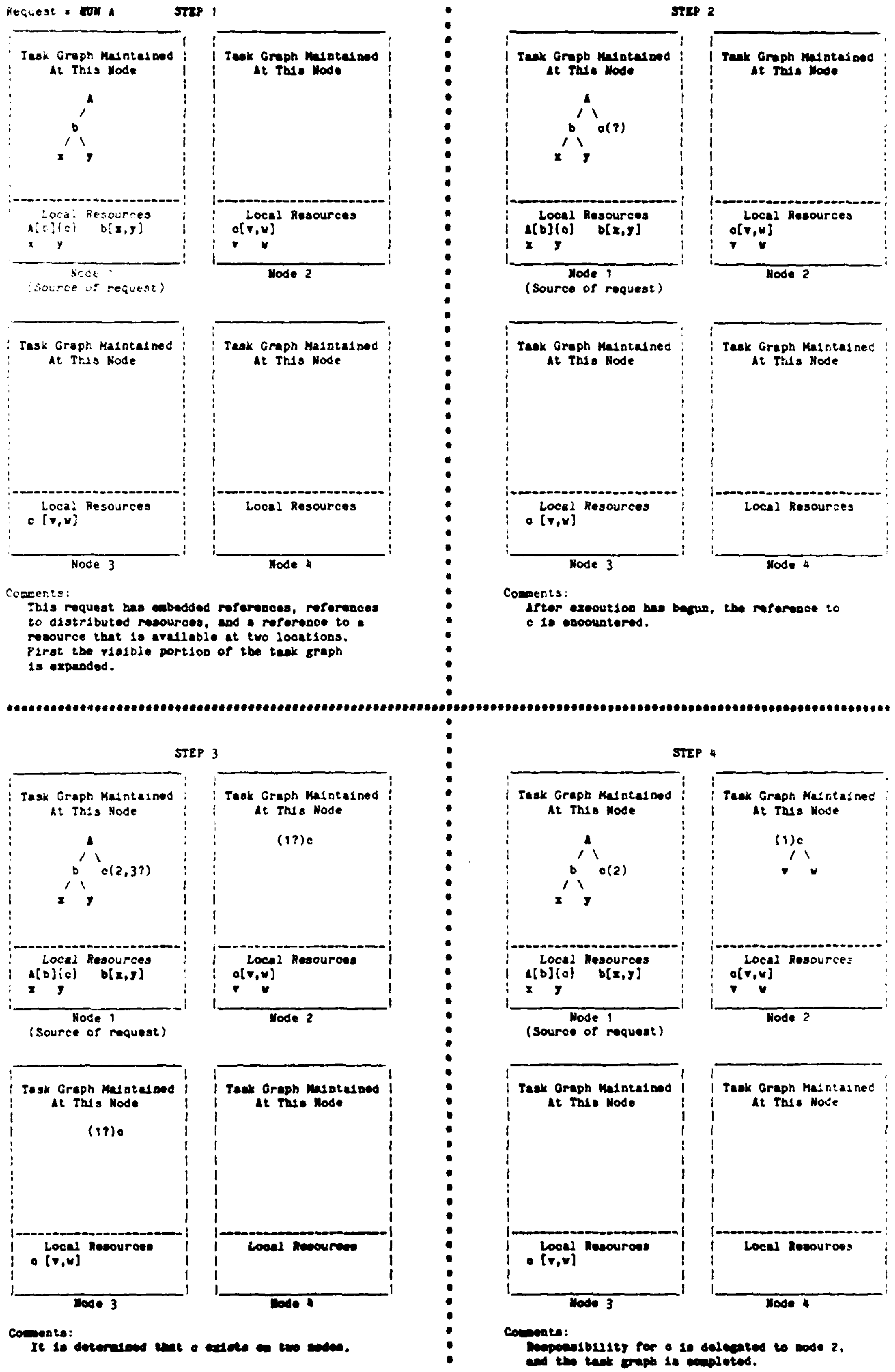
STEP 3 Comments:
After further execution, a reference to y is discovered and entered into the task graph. The task graph is now complete.

Figure 27. Example 7

Multiple steps, though, are required to construct the task graph because not all of the resources are visible and thus cannot be identified until after execution has progressed to the point where the reference is encountered. Example 8 (Figure 28) is slightly more complex with resources spread over multiple nodes. Again multiple steps are required since parts of the task graph cannot be constructed until their references are observed. In addition since resources are distributed on different nodes, negotiation must occur.

The last three examples demonstrate three different techniques for storing task graphs. In each example, the same work request is utilized. This request has all visible references to resources distributed over multiple nodes. In the first eight examples and example 9 (Figure 29), the parts of the overall task graph are stored on the nodes executing their processes. In addition, each subgraph contains a small portion of information linking it to the rest of the overall task graph. Example 10 (Figure 30) maintains these subgraphs and in addition retains a complete task graph at the source node. Finally, example 11 (Figure 31) maintains complete task graphs at all nodes where processing occurs. The motivation for the last two techniques in which a large amount of redundant information is maintained is to enhance the ability to recover from failures.

Now that we have taken a look at the construction of task graphs in a broad sense, let us examine the details of the task of processing a work request. This is illustrated in two figures. Figure 32 outlines the basic steps involved in work request processing. Finally, Figure 33 depicts the steps involved in processing a specific work request. In this case, the work request is the same as that from example 6 (c.f., Figure 26).



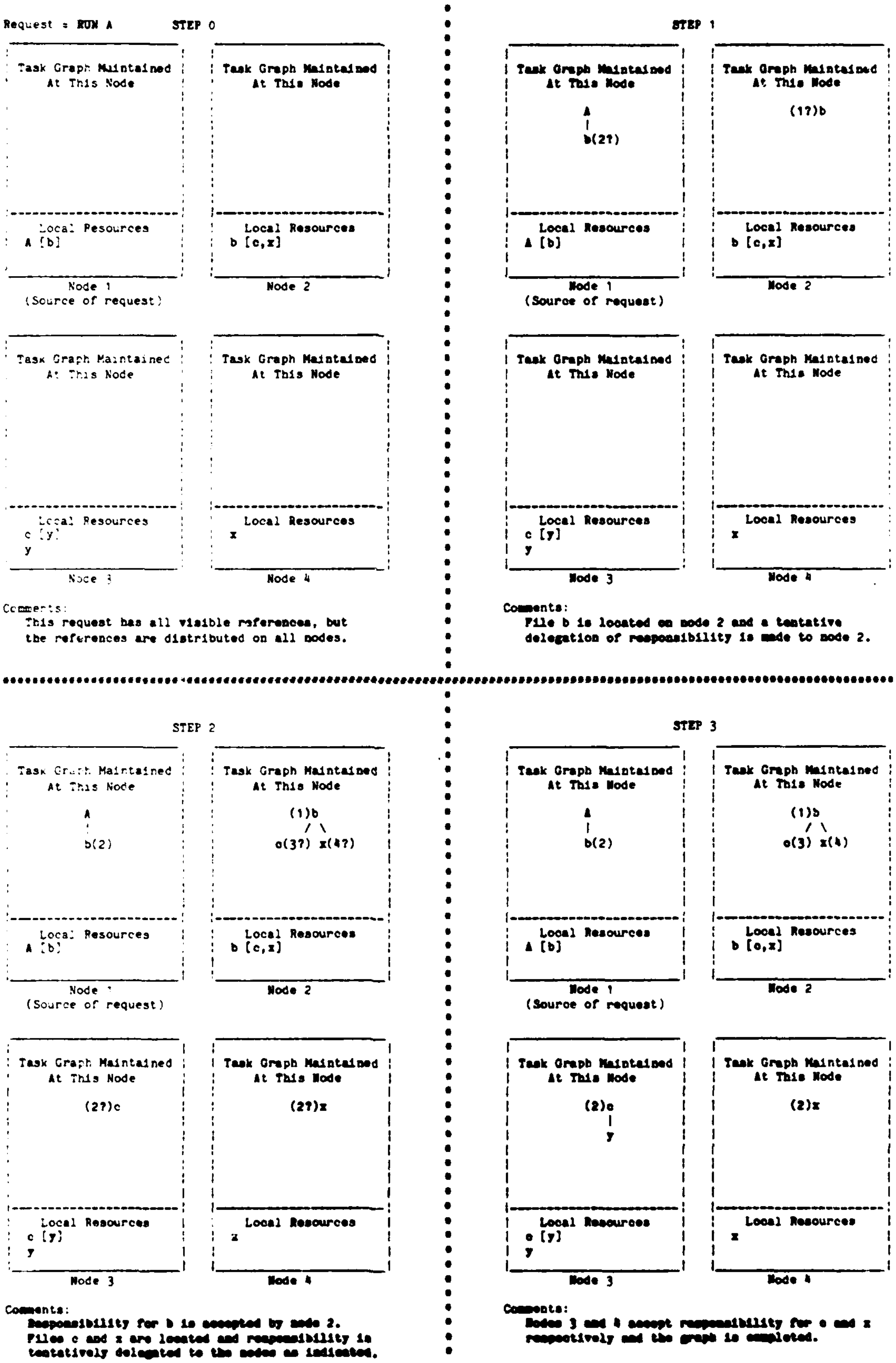
Step 1 Comments:
This request has embedded references, references to distributed resources, and a reference to a resource that is available at two locations. First the visible portion of the task graph is expanded.

Step 2 Comments:
After execution has begun, the reference to c is encountered.

Step 3 Comments:
It is determined that c exists on two nodes.

Step 4 Comments:
Responsibility for c is delegated to node 2, and the task graph is completed.

Figure 28. Example 8



Step 0 Comments:
This request has all visible references, but the references are distributed on all nodes.

Step 1 Comments:
File b is located on node 2 and a tentative delegation of responsibility is made to node 2.

Step 2 Comments:
Responsibility for b is accepted by node 2. Files c and x are located and responsibility is tentatively delegated to the nodes as indicated.

Step 3 Comments:
Nodes 3 and 4 accept responsibility for c and x respectively and the graph is completed.

Figure 29. Example 9

Request = RUN A

STEP 1

```
 ------------------------    ------------------------
| Task Graph Maintained |   | Task Graph Maintained |
|     At This Node      |   |     At This Node      |
|                       |   |                       |
|          A            |   |        (1?)b          |
|          |            |   |                       |
|         b(2?)         |   |                       |
|                       |   |                       |
|-----------------------|   |-----------------------|
|    Local Resources    |   |    Local Resources    |
|  A [b]                |   |  b [c,x]              |
|                       |   |                       |
 ------------------------    ------------------------
         Node 1                      Node 2
  (Source of request)

 ------------------------    ------------------------
| Task Graph Maintained |   | Task Graph Maintained |
|     At This Node      |   |     At This Node      |
|                       |   |                       |
|                       |   |                       |
|-----------------------|   |-----------------------|
|    Local Resources    |   |    Local Resources    |
|  c [y]                |   |  x                    |
|  y                    |   |                       |
 ------------------------    ------------------------
         Node 3                      Node 4
```

Comments:
In this example a complete copy of the task graph is to be maintained at the node receiving the request. File b is located on node 2 and a tentative delegation of responsibility is made to node 2.

STEP 2

```
 ------------------------    ------------------------
| Task Graph Maintained |   | Task Graph Maintained |
|     At This Node      |   |     At This Node      |
|                       |   |                       |
|          A            |   |         (1)b          |
|          |            |   |         /  \          |
|         b(2)          |   |     c(3?) x(4?)       |
|                       |   |                       |
|-----------------------|   |-----------------------|
|    Local Resources    |   |    Local Resources    |
|  A [b]                |   |  b [c,x]              |
|                       |   |                       |
 ------------------------    ------------------------
         Node 1                      Node 2
  (Source of request)

 ------------------------    ------------------------
| Task Graph Maintained |   | Task Graph Maintained |
|     At This Node      |   |     At This Node      |
|                       |   |                       |
|        (2?)c          |   |        (2?)x          |
|                       |   |                       |
|-----------------------|   |-----------------------|
|    Local Resources    |   |    Local Resources    |
|  c [y]                |   |  x                    |
|  y                    |   |                       |
 ------------------------    ------------------------
         Node 3                      Node 4
```

Comments:
Responsibility for b is accepted by node 2. Files c and x are located and responsibility is tentatively delegated to the nodes as indicated.

STEP 3

```
 ------------------------    ------------------------
| Task Graph Maintained |   | Task Graph Maintained |
|     At This Node      |   |     At This Node      |
|                       |   |                       |
|          A            |   |         (1)b          |
|          |            |   |         /  \          |
|         b(2)          |   |       c(3) x(4)       |
|         /  \          |   |                       |
|      c(3) x(4)        |   |                       |
|                       |   |                       |
|-----------------------|   |-----------------------|
|    Local Resources    |   |    Local Resources    |
|  A [b]                |   |  b [c,x]              |
|                       |   |                       |
 ------------------------    ------------------------
         Node                        Node 2
      [illegible] request

 ------------------------    ------------------------
| [illegible] Maintained |  | Task Graph Maintained |
|     [illegible]       |   |     At This Node      |
|                       |   |                       |
|                       |   |         (2)x          |
|                       |   |                       |
|-----------------------|   |-----------------------|
|     [illegible]       |   |    Local Resources    |
|                       |   |  x                    |
|                       |   |                       |
 ------------------------    ------------------------
                                     Node 4
```

[illegible] responsibility for c and x [illegible] noted in the copy of [illegible]

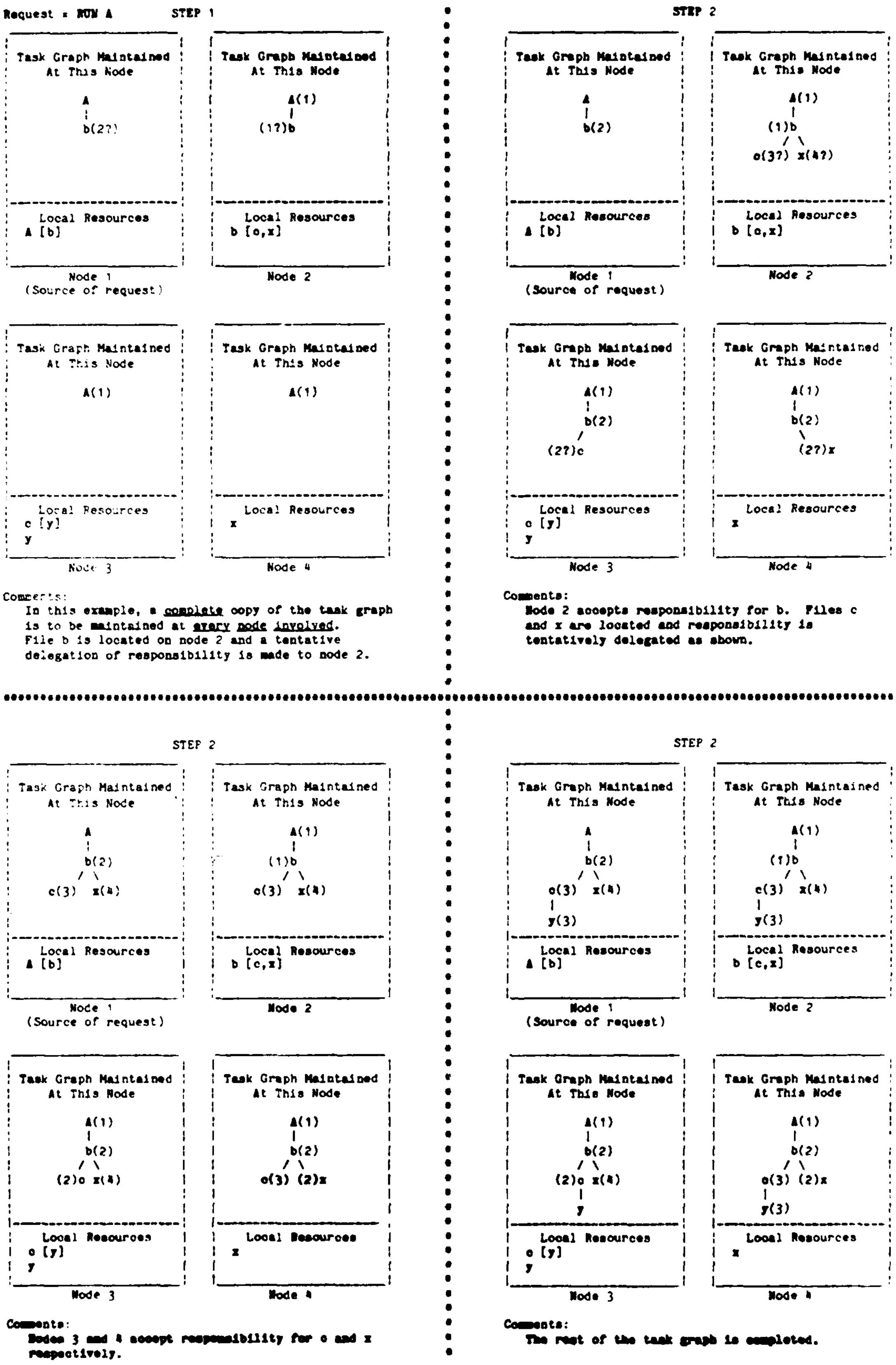
STEP 4

```
 ------------------------    ------------------------
| Task Graph Maintained |   | Task Graph Maintained |
|     At This Node      |   |     At This Node      |
|                       |   |                       |
|          A            |   |         (1)b          |
|          |            |   |         /  \          |
|         b(2)          |   |       c(3) x(4)       |
|         /  \          |   |                       |
|      c(3) x(4)        |   |                       |
|       |               |   |                       |
|      y(3)             |   |                       |
|-----------------------|   |-----------------------|
|    Local Resources    |   |    Local Resources    |
|  A [b]                |   |  b [c,x]              |
|                       |   |                       |
 ------------------------    ------------------------
         Node 1                      Node 2
  (Source of request)

 ------------------------    ------------------------
| Task Graph Maintained |   | Task Graph Maintained |
|     At This Node      |   |     At This Node      |
|                       |   |                       |
|         (2)c          |   |         (2)x          |
|          |            |   |                       |
|          y            |   |                       |
|                       |   |                       |
|-----------------------|   |-----------------------|
|    Local Resources    |   |    Local Resources    |
|  c [y]                |   |  x                    |
|  y                    |   |                       |
 ------------------------    ------------------------
         Node 3                      Node 4
```

Comments:
The rest of the task graph is completed.

Figure 30. Example 10



Comments:
In this example, a complete copy of the task graph is to be maintained at every node involved. File b is located on node 2 and a tentative delegation of responsibility is made to node 2.

Comments:
Node 2 accepts responsibility for b. Files c and x are located and responsibility is tentatively delegated as shown.

Comments:
Nodes 3 and 4 accept responsibility for c and x respectively.

Comments:
The rest of the task graph is completed.

Figure 31. Example 11

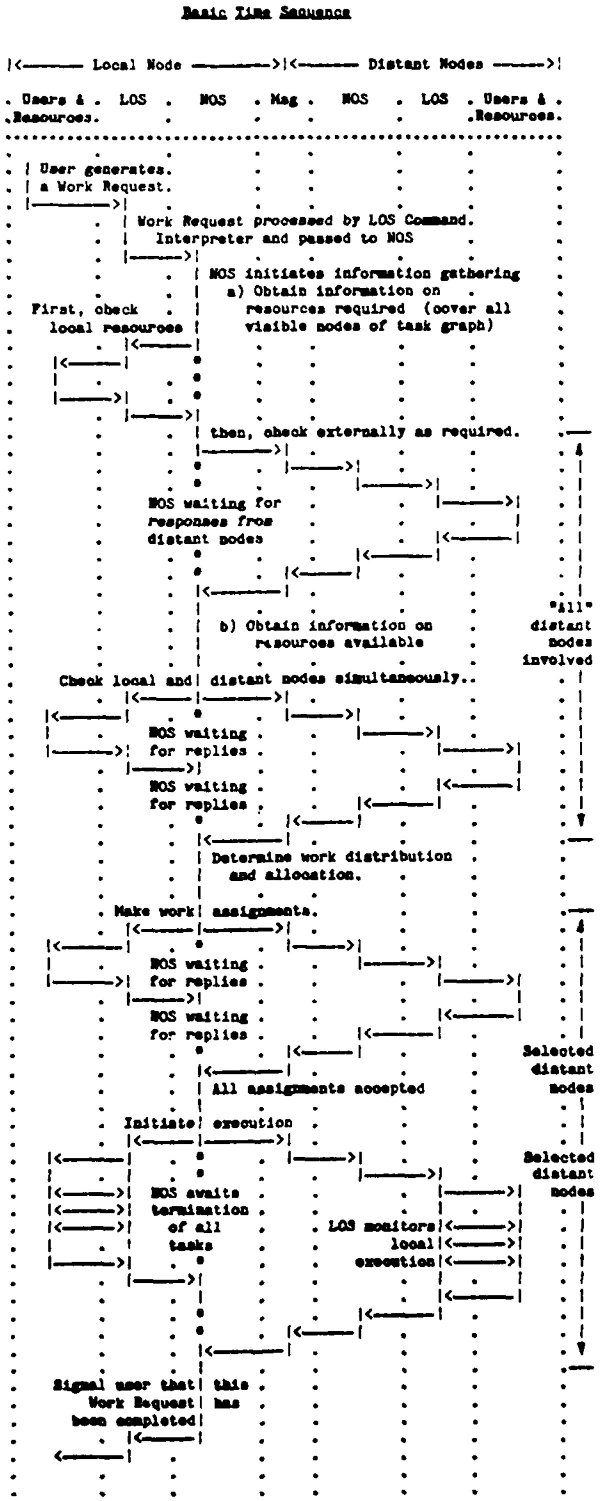

Basic Time Sequence
Local Node
Distant Nodes
Users & Resources
LOS
NOS
Msg
NOS
LOS
Users & Resources
User generates a Work Request.
Work Request processed by LOS Command. Interpreter and passed to NOS
NOS initiates information gathering
a) Obtain information on resources required (cover all visible nodes of task graph)
First, check local resources
then, check externally as required.
NOS waiting for responses from distant nodes
"All" distant nodes involved
b) Obtain information on resources available
Check local and distant nodes simultaneously..
NOS waiting for replies
NOS waiting for replies
Determine work distribution and allocation.
Make work assignments.
NOS waiting for replies
NOS waiting for replies
Selected distant nodes
All assignments accepted
Initiate execution
Selected distant nodes
NOS awaits termination of all tasks
LOS monitors local execution
Signal user that this Work Request has been completed

Figure 32. Basic Steps in Work Request Processing

Situation Same As Example A (Figure 26)

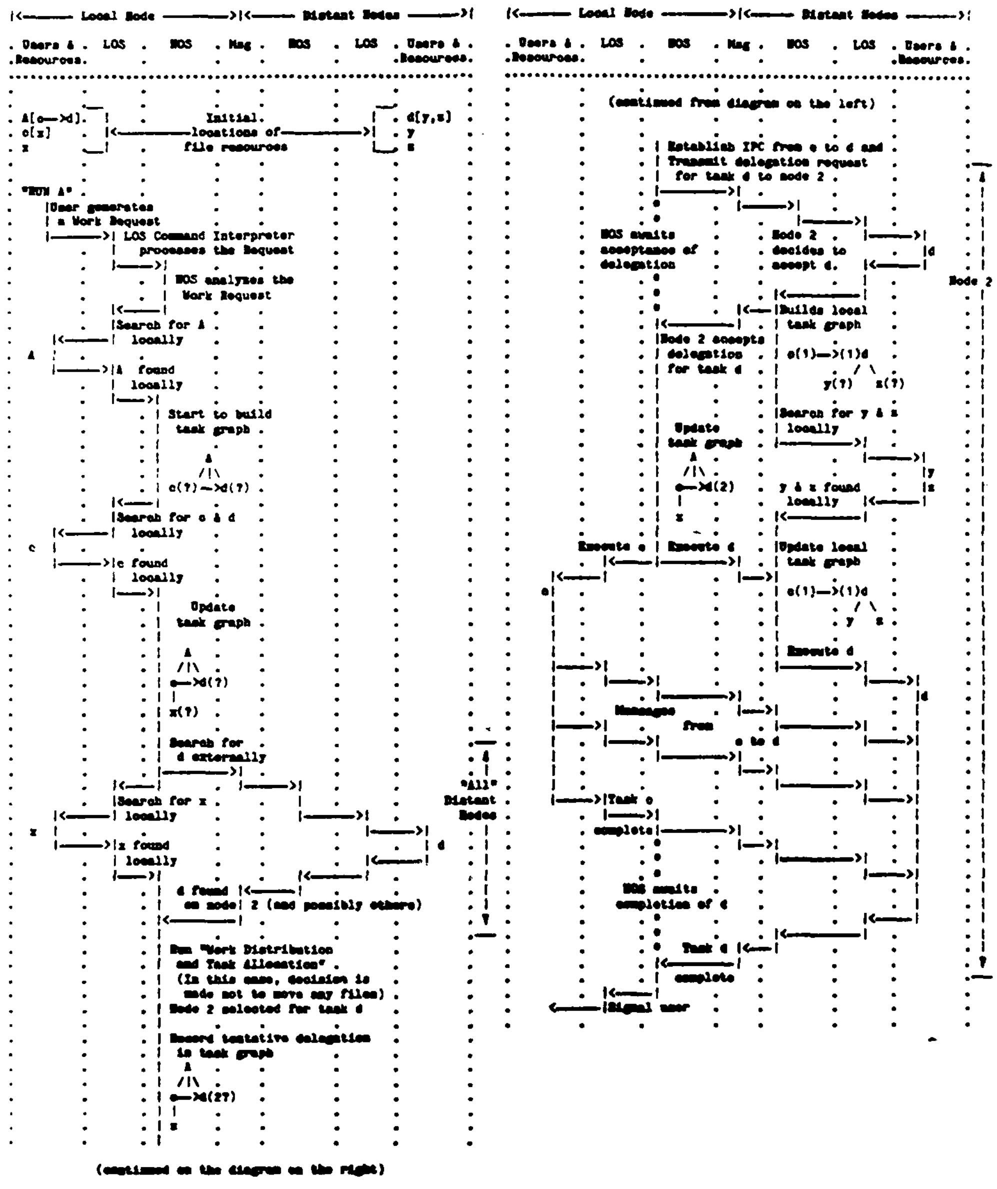

|<——— Local Node ———>|<——— Distant Nodes ———>|
Users & Resources
LOS
NOS
Msg
NOS
LOS
Users & Resources
A[c—>d]
c[x]
x
Initial locations of file resources
d[y,z]
y
z
"RUN A"
User generates a Work Request
LOS Command Interpreter processes the Request
NOS analyzes the Work Request
Search for A locally
A
A found locally
Start to build task graph
A
/|\
c(?)—>d(?)
Search for c & d locally
c
c found locally
Update task graph
A
/|\
c—>d(?)
|
x(?)
Search for d externally
"All" Distant Nodes
Search for x locally
x
x found locally
d
d found on node 2 (and possibly others)
Run "Work Distribution and Task Allocation".
(In this case, decision is made not to move any files)
Node 2 selected for task d
Record tentative delegation in task graph
A
/|\
c—>d(2?)
|
x
(continued on the diagram on the right)
(continued from diagram on the left)
Establish IPC from c to d and Transmit delegation request for task d to node 2.
NOS awaits acceptance of delegation
Node 2 decides to accept d.
d
Node 2
Builds local task graph
c(1)—>(1)d
/ \
y(?) z(?)
Node 2 accepts delegation for task d
Search for y & z locally
Update task graph
A
/|\
c—>d(2)
|
x
y
z
y & z found locally
Execute c
Execute d
Update local task graph
c
c(1)—>(1)d
/ \
y z
Execute d
d
Messages from c to d
Task c complete
NOS awaits completion of d
Task d complete
Signal user

Figure 33. An Example of Work Request Processing

SECTION 8

MODELS OF CONTROL

In this section, we demonstrate how both existing and proposed models of control fit into the classification scheme described in Section 7. With the exception of the first model, these controls are designed to service work requests that specify multiple concurrent communicating processes. The first model considers work requests that involve only a single process.

8.1 ARAMIS

A decentralized operating system model for the ARAMIS Distributed Computer System is described in [Caba79a,b]. A brief outline of how this model fits into the classification scheme of Section 7 is provided by Table 3.

8.1.1 Architecture

The ARAMIS Distributed Computer System consists of two types of machines, hosts and managers. Users are connected to hosts which in turn are connected to managers. The managers are connected to each other in a virtual ring. Execution of work requests is provided by the hosts while control decisions are made by the managers.

8.1.2 Work Requests

This system is designed to handle a work request that is less sophisticated than those handled by the other systms described in this section. The work request must specify only a single process or task and the list of resources (sharable and nonsharable) required by that task.

8.1.3 The Control Model

Control of the system is accomplished through the managers. Each manager maintains a data structure called the resource state table (RST) which contains state information for every resource available on the system. To insure that these redundant copies remain consistent, two vectors are utilized. The control vector (CV) cycles around the virtual ring. Only the manager possessing the CV is permitted to allocate and deallocate resources. Upon completing this work, a manager can pass the CV along. In addition, modifications made to the RST (information describing the allocation and deallocation of files) are passed along to the other managers on the virtual ring in the form of an update vector (UPV).

Table 3. The Decentralized Control Model of the ARAMIS Distributed Computer System

TASK GRAPH CONSTRUCTION:

Who builds the task graph?
A manager on each node builds the task graph for the work requests arriving at that node.

What is the nature of the task graph?
A single structure.

Where is the task graph stored?
On the node initially receiving and analyzing the work request and the node where execution of the task occurs.

When is the task graph built?
Completely prior to execution.

RESOURCE AVAILABILITY INFORMATION:

Who maintains this information?
All nodes maintain common information.

Where is the information maintained?
In multiple redundant copies.

ALLOCATION OF RESOURCES:

How is concurrency control provided?
Resources are locked before the work distribution decision is made.

PROCESS INITIATION:

How is responsibility distributed?
Each node has a manager. The node initially receiving and analyzing the work request retains enough information to restart the task if the execution node dies.

How is refusal of a request to execute a process by a node handled?
This possibility is not discussed.

PROCESS MONITORING:

What type of interprocess communication is provided?
IPC is not supported.

How are task graphs resulting from additional work requests handled?
Additional requests cannot occur.

When a work request arrives at a host, it is passed along to the local manager to which the host is connected. This manager is in charge of resource allocation and task routing. It first identifies the resources that are needed and allocates sharable resources. After the CV has arrived and various algorithms insuring mutual exclusion and the prevention of deadlocks have been executed, the nonsharable resources are allocated. Next the optimal site for execution of the task is determined taking into account the burden various choices place on the communication system. Finally, the information concerning the allocation of resources is transmitted in the form of a UPV, and the information describing the task routing is sent to the hosts needing the information.

8.1.4 Conclusion

This model represents a simplified approach to the control problem. All nodes are provided with a complete global view of the system via their copy of the RST. Modifications to the state are carefully controlled by permitting only one manager at a time to change this information. The capability to perform modifications on the RST is passed around the virtual ring in the form of the CV.

8.2 MEDUSA

Medusa [Oust80a,b] is a distributed operating system for the Carnegie-Mellon Cm* multimicroprocessor. This system differs from an FDPS in that it allows multiple nodes to share primary memory. Table 4 describes how this control model fits into the classification scheme of Section 7.

8.2.1 Architecture

Cm* consists of a number of relatively independent processors or computer modules (Cm) and a number of communication controllers (Kmap). The Cm's are arranged in clusters with a Kmap presiding over each cluster. A switch, Slocal, connects a Cm with the interprocessor communication structure. Each Slocal contains tables that allow it to decide on each memory reference whether to access local memory or pass the reference along to the Kmap to locate the desired information in either the local cluster or a distant cluster. Thus, any processor can access the memory of any other processor. It must be kept in mind, though, that a substantial time delay results from accessing the memory of distant processors.

Table 4. The Medusa Control Model

TASK GRAPH CONSTRUCTION:

Who builds the task graph?
The node containing an activation of the task force manager.

What is the nature of the task graph?
Multiple structures (the task force control block is stored in the SDL and the activity control block is stored in the PDLs).

Where is the task graph stored?
Multiple nodes.

When is the task graph built?
Completely prior to execution.

RESOURCE AVAILABILITY INFORMATION:

Who maintains this information?
A number of utilities each realized as a task force.

Where is the information maintained?
In a shared data structure.

ALLOCATION OF RESOURCES:

How is concurrency control provided?
By means of locks.

PROCESS INITIATION:

How is responsibility distributed?
The task force manager keeps overall control, but other special managers are available to provide specific services.

How is refusal of a request to execute a process by a node handled?
This is not discussed in the literature.

PROCESS MONITORING:

What type of interprocess communication is provided?
Unsynchronized communication.

How are task graphs resulting from additional work requests handled?
It is not clear if additional work can be requested.

8.2.2 Work Requests

Work requests are used to describe task forces. A task force consists of a number of relatively independent communicating processes capable of concurrent execution that are working toward the solution of some task. Interprocess communication is accomplished via pipes which differ slightly from those found in UNIX [Ritc78]. There are two unique features found in these pipes: 1) they insure that only whole messages are read, and 2) they identify the sender of the message to the receiver.

In addition to processes and pipes, a task force contains a shared descriptor list (SDL) and a number of private descriptor lists (PDL). These structures contain descriptors which are basically capabilities for certain system objects. There is only one SDL per task force. This provides access to objects that are shared among all processes of a task force. For each process, there is a PDL which provides access to private objects. Thus, the significant feature of the task force concept is the capability to directly share objects by means of the SDL.

8.2.3 The Control Model

The distributed control is composed of a series of five utilities each of which is implemented as a task force. The five utilities are as follows:

1. Memory Manager: allocates primary memory and aids the Kmap in descriptor list manipulation.
2. File System: acts as a controller for all I/O devices of the system and implements a hierarchical file system.
3. Task Force Manager: creates, schedules, and deletes task forces and the processes that comprise task forces.
4. Exception Reporter: communicates information about unusual occurrences to those processes that need to know this information.
5. Debugger/Tracer: holds symbol table and performance measurement information for all utilities and provides facilities for on-line debugging of the system and gathering of performance data.

Communication between user processes and utilities is accomplished by means of pipes. There is one pipe for each utility. Access to these pipes is provided by the utility descriptor list (UDL) which is present on all nodes. A process utilizes this structure to locate the proper pipe into which a message for a particular utility is to be placed.

8.2.4 Conclusion

Medusa introduces two features that are pertinent to this discussion. These are the concept of a task force and the concept of sharing primary memory. A task force provides concurrent communicating processes to solve a common task. In addition to communicating by means of messages, processes are permitted to share data. The idea of shared memory is also seen in the hardware by the ability to directly reference memory on distant processors.

8.3 CNET

CNET [Smit79, Smit80] is a distributed problem solver consisting of a collection of loosely coupled knowledge sources located on a number of distinct processors. Table 5 depicts how this model fits into the classification scheme of Section 7.

8.3.1 Architecture

The system is intended for use on a network of loosely coupled asynchronous processors. Communication between nodes is realized through broadcast messages.

8.3.2 Work Requests

Applications for CNET can potentially take the form of cooperating processes. An individual work request specifies the work that must be accomplished. Depending upon decisions of the control, a task may be divided into subtasks, and the subtasks may be further divided.

8.3.3 The Control Model

CNET utilizes a hierarchical form of control for each task. At the top level is the manager for the task that is described in the original work request. This manager attempts to find a suitable contractor to execute the task. This is accomplished by means of a negotiation that begins with a message from the manager. This message can take the form of a general broadcast, a limited broadcast, or a point-to-point announcement. The contents of the message include an eligibility specification (a list of criteria required of a node to execute the task), a task abstraction (a brief description of the task), a bid specification (describes the expected form of a bid from a possible contractor), and an expiration time (describes the time period that the announcement is valid). A general broadcast is utilized when the manager has no knowledge concerning the nodes capable of executing the task. A limited

Table 5. The CNET Control Model

TASK GRAPH CONSTRUCTION:

Who builds the task graph?
Multiple nodes.

What is the nature of the task graph?
Multiple structures each consisting of a subgraph.

Where is the task graph stored?
Multiple nodes.

When is the task graph built?
Piecemeal.

RESOURCE AVAILABILITY INFORMATION:

Who maintains this information?
Each node maintains information about its own resources.

Where is the information maintained?
Separate pieces of information concerning a particular resource type may be kept on different nodes.

ALLOCATION OF RESOURCES:

How is concurrency control provided?
Resources are locked before the work distribution decision is made.

PROCESS INITIATION:

How is responsibility distributed?
There is a hierarchy of responsibility.

How is refusal of a request to execute a process by a node handled?
Once a contract is made it is binding.

PROCESS MONITORING:

What type of interprocess communication is provided?
Not specified.

How are task graphs resulting from additional work requests handled?
The new task graph is kept separate.

broadcast can be utilized when the manager knows a specific group of nodes is capable of executing the task. Finally, a point-to-point announcement is made when the manager knows about the availability of a single suitable node. This knowledge is obtained from idle nodes that broadcast messages indicating their availability.

The manager sends these messages and waits for the arrival of bids from possible contractors. When the bids arrive, they are examined in order to determine a choice for the task assignment. All bids are binding so the manager can make a choice with confidence that a chosen node will accept the task. Once a node is chosen, the contract is awarded and the chosen node becomes known as a contractor. The contractor may further divide the task and utilize other contractors for the various pieces. Thus, a node can act both as a manager and a contractor.

A contractor provides the manager with reports that contain information concerning partial execution (interim report) or completion (final report). A report contains a result description that specifies execution results. A manager has complete authority over a contractor and thus may terminate contracts at any time with a termination message. This terminates execution of a contract and all outstanding subcontracts.

8.3.4 Conclusion

CNET utilizes a hierarchical control scheme with a manager supervising the work of possibly multiple contractors working to solve a given task. A manager locates contractors by broadcasting an announcement for bids. It then waits for the bids from the contractors to arrive. After this negotiation phase, a bid is accepted, a contract is awarded, and execution of the task is begun. The manager can terminate execution of a task at any time and is the recipient of interim and final reports from the contractors.

8.4 THE XFDPS SERIES OF MODELS

In Section 7, a list of design alternatives for an FDPS executive control is presented (See Table 2). The rest of this section is devoted to the presentation of a series of control models designed by this research team by choosing among these alternatives. Each of the models is referred to as XFDPS.i where i is an identifying numeral. It is neither possible nor practical to present all possible models for an FDPS executive control.

Therefore, only a few models are investigated. The models were chosen by selecting a collection of design alternatives which were both logical and provided significant distinction among the various models.

The models are described in such a manner as to give the reader a feeling for the overall control strategy. A more complete comparison of the models can be obtained through tables 6 through 8 which contain a list of design alternatives for each model.

8.4.1 Architecture

An FDPS is composed of a multiplicity of independent processors physically connected by a network providing communication by means of a two-party protocol. There is no sharing of primary memory, and, thus, the processors are considered to be loosely coupled. The processors operate in an autonomous but cooperative manner. Therefore, it is the responsibility of the control to insure that there is a unification of operation in the system.

8.4.2 Work Requests

Work requests describe concurrent communicating processes ana are assumed to provide the functionality available with the command language described in Figure 10.

8.4.3 XFDPS.1

The XFDPS.1 model [Sapo80] (see Table 6 for a characterization of this model and Figure 34 for a view of the model's components) is a distributed and decentralized control model that is designed to shield the user from the system. In other words, it provides the system transparency that is fundamental to the FDPS definition. It is designed to encapsulate each processor's local operating system as advocated by Kimbleton [Kimb76]. This is the meta-system approach to implementing distributed operating systems discussed above and has been practiced in several systems including ADAPT [Peeb80]. The XFDPS.1 model is composed of a set of cooperating processes called managers and is similar in this respect to Medusa [Oust80] and ADAPT [Peeb80]. Each manager is designed to control a subset of the system's resources (logical and physical).

Each manager requires reliable message communication with the other managers in order to perform its responsibilities. The XFDPS.1 model does not assume the presence of any particular interconnection of processors or for that matter any particular technique of message communication. This means

Table 6. The XFDPS.1 Control Model

TASK GRAPH CONSTRUCTION:

Who builds the task graph?
The source node.

What is the nature of the task graph?
Multiple structures each consisting of a subgraph with one copy of the complete task graph.

Where is the task graph stored?
Multiple nodes.

When is the task graph built?
Completely prior to execution.

RESOURCE AVAILABILITY INFORMATION:

Who maintains this information?
Each node maintains information about its own resources.

Where is the information maintained?
At the node which contains the resource.

ALLOCATION OF RESOURCES:

How is concurrency control provided?
Reservations are used prior to a work distribution decision and then allocated by a lock.

PROCESS INITIATION:

How is responsibility distributed?
There is a hierarchy of responsibility.

How is refusal of a request to execute a process by a node handled?
After repeated attempts, the request is abandoned.

PROCESS MONITORING:

What type of interprocess communication is provided?
Unsynchronized communication.

How are task graphs resulting from additional work requests handled?
The new task graph is kept separate.

AD-A104 379 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
DISTRIBUTED AND DECENTRALIZED CONTROL IN FULLY DISTRIBUTED PROC--ETC(U)
JUN 81 P H ENSLOW, T G SAPONAS F30602-78-C-0120
UNCLASSIFIED GIT-ICS-81/02 RADC-TR-81-127 NL

2 of 2
AD
A104379

END
DATE
FILMED
0 81
DTIC

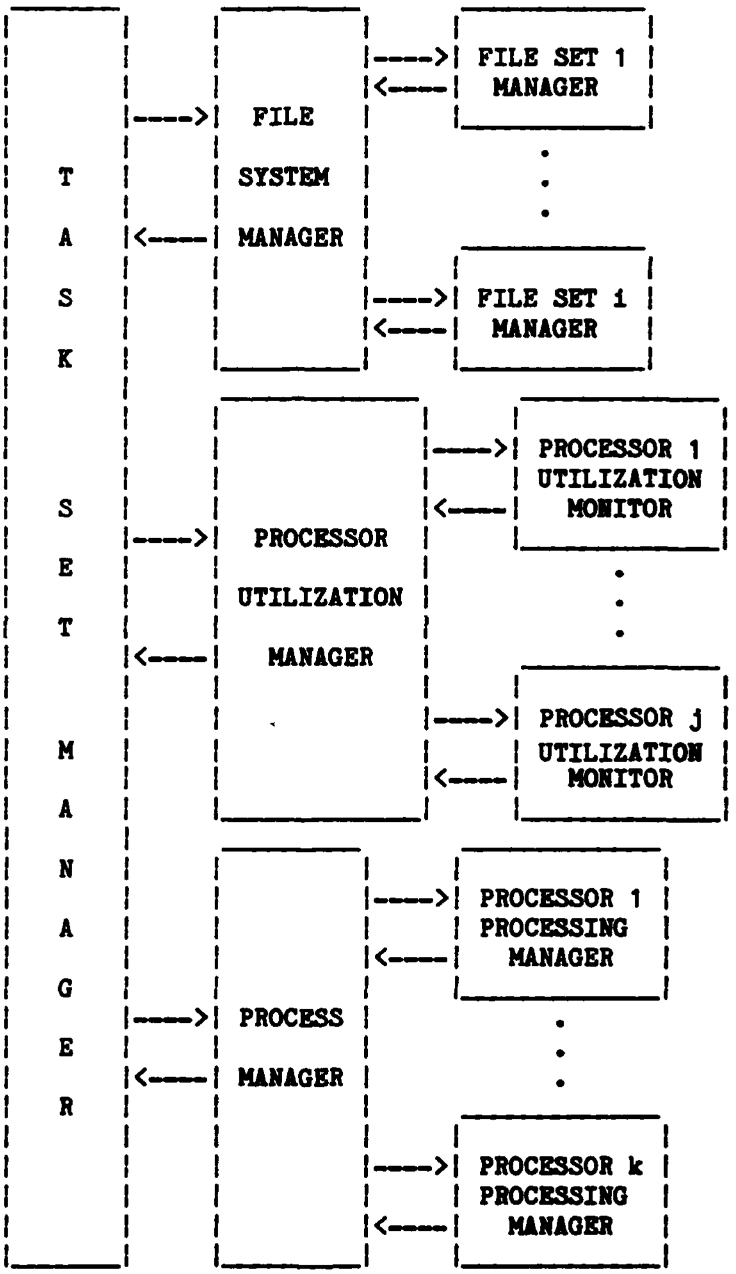

TASK SET MANAGER
FILE SYSTEM MANAGER
FILE SET 1 MANAGER
FILE SET i MANAGER
PROCESSOR UTILIZATION MANAGER
PROCESSOR 1 UTILIZATION MONITOR
PROCESSOR j UTILIZATION MONITOR
PROCESS MANAGER
PROCESSOR 1 PROCESSING MANAGER
PROCESSOR k PROCESSING MANAGER

Figure 34. The XFDPS.1 Control Model

that the model is applicable to systems that are interconnected in a variety of ways including loops, stars, regular networks, irregular networks, or fully interconnected networks [Ande75] and utilizing various message communication techniques including the ISO model [Bach78, Desj78] and Ethernet [Metc76].

The XFDPS.1 model is composed of several types of processes called managers which are responsible for various aspects of the control problem. These managers include the Task Set Manager, the File System Manager, the Processor Utilization Manager, and the Process Manager.

8.4.3.1 Task Set Manager

The Task Set Manager is responsible for handling work requests arriving from eitner users or active processes. A Task Set Manager is assigned to every work request. It must first identify the tasks comprising the Task Set which are needed to satisfy the work request and then communicate with the File System Manager to obtain information concerning the availability of files. The Processor Utilization Manager is also consulted in order to determine the relative utilization of the processors. Using the information acquired in this manner, a work allocation decision is made that results in the assignment of tasks to processors. This decision involves an optimization problem similar in many respects to that discussed by Morgan [Morg77].

The second phase of the Task Set Manager's responsibility concerns carrying out the decision arrived at in the first phase. This again involves communication with the File System Manager to allocate needed files and to deallocate these files when they are no longer needed. In addition, communication is required with the Process Manager which activates the processes and observes when these processes have terminated. The last act of the Task Set Manager is to inform the original requester as to the completion status of the request. In doing so it will either indicate that it was successful in completing the request or provide a description concerning why the request could not be completed.

8.4.3.2 File System Manager

The File System Manager is responsible for maintaining the file system for the entire FDPS. Instances of the File System Manager are found on all processors. Management of the file system is achieved through communication among these instances of the File System Manager.

The implication of this design is that several requests to the file system can be acted upon simultaneously provided these requests arrive at different processors. These requests may either elicit availability information or ask that the file status information be updated (i.e., making a reservation, placing a lock, or releasing a lock on a file). This simultaneity is in marked contrast to the resource allocation found in the ARAMIS Distributed Computer System [Caba79a,b] in which all nodes possess a Resource State Table containing the state of all resources in the system. This system only permits resource allocation by at most one node at any one time.

In the XFDPS.1 model, the file system is divided into several disjoint sets. The design of the control does not restrict how this division is realized. For example, these sets can be defined by processor boundaries. For each set, there is a separate manager called a File Set Manager. In order to perform its management duties, the File System Manager must communicate with each File Set Manager.

The File System Manager handles three types of requests, all originating from the Task Set Manager. The first type of request is for availability information concerning a collection of files. The File System Manager converts this request into a series of requests concerning individual files and presents these requests to the File Set Managers. The File System Manager waits for responses from all File Set Managers before returning its response. A File Set Manager will return an indication of the file's availability. If a file is available, the File Set Manager will reserve the file for the Task Set from which the request originated. This reservation remains effective for a limited period of time, and it is the responsibility of the Task Set Manager to confirm the reservation before its effectiveness has expired.

The second request that can be made to the File System Manager concerns the allocation of a series of files. Again this request is converted into a number of requests concerning the reservations of individual files and is sent to specific File Set Managers which in turn perform the necessary locking of the files.

Finally, the File System Manager can receive requests for the deallocation of files. These requests are handled in a manner similar to allocation requests and result in the release of locks or reservations on specific files.

8.4.3.3 Processor Utilization Manager

Another type of process found in the control is the Process Utilization Manager. Instances of this manager are replicated on all processors. The main function of the Process Utilization Manager is the maintenance of a data base of processor utilization information for the processors comprising the FDPS. The information in this data base is not intended to be complete and accurate but rather is designed to provide the work assignment algorithm in the Task Manager with an estimate of the utilization of the processors in the system.

The Processor Utilization Manager obtains the information needed to update its data base from periodic messages directed to it from Processor Utilization Monitors located on each processor. These processes monitor the utilization of the processor in which they are located and issue periodic messages reporting their findings. If a Processor Utilization Manager does not receive a report from a Processor Utilization Monitor within a certain period of time, a message from the Manager is sent to the Monitor asking for an immediate response concerning the processor's state. If a response to this request is not received within a certain time period, it is assumed the processor is lost, and the Processor Utilization Manager updates its data base to reflect this. This will prevent the Task Set Manager from attempting to assign processes to a processor that has apparently been lost.

8.4.3.4 Process Manager

The last process type found in the control is the Process Manager. A Process Manager is activated for each Task Set Manager. This process accepts requests from the Task Set Manager for the activation of processes for the Task Set. The Process Manager identifies which processors are to receive processes. It then issues requests to Processing Managers on each processor. Each Processing Manager is responsible for controlling the processes assigned to its processor.

In addition to assigning processes and waiting for the notification of their termination, the Process Manager is responsible for providing interprocess communication between executing processes. In this model, interprocess communication is provided by means of ports [Balz71, Have78, Suns77, Zuck77]. A port provides a common location where communicating processes can either send or fetch messages without knowing about the other's location. Buffer space is also required in order to allow the communicating processes to operate as independently as possible. This type of interprocess communication is similar to the stream communication utilized in TRIX [Ward80]. The Process Manager must therefore decide where a buffer for the port resides and then provide the necessary linkages within the communicating processes in order for them to address the port.

8.4.3.5 Conclusion

The fundamental philosophy of the XFDPS.1 model is that the control over logical and physical resources must be distributed among various processes or managers. The reason for taking this approach is to provide better utilization of system resources by making use of the inherent parallelism found in distributed processing systems.

8.4.4 XFDPS.2

XFDPS.2 is a variation of model XFDPS.1. The main difference between the two models exists in the technique used to construct the task graph. A complete outline of the characteristics of XFDPS.2 is found in Table 7.

The construction of task graphs in XFDPS.2 is performed by multiple nodes resulting in a task graph.that consists of multiple structures each of which is a subgraph of the complete task graph. The overall strategy works as follows. After a work request arrives at a particular node, work on constructing a task graph is begun. When a node is chosen to perform part of a task graph, responsibility for that portion of the task graph is given to a control component on that node. This component will maintain that portion of the task graph and in so doing may also choose other nodes to perform part of the work that the subgraph represents.

Thus, there are two main differences between XFDPS.2 and XFDPS.1: 1) the task graph is not maintained in one location but rather on multiple nodes, and 2) this construction is performed in a piecemeal fashion in XFDPS.2. This means that the components of XFDPS.2 possess greater independence than those of XFDPS.1.

8.4.5 XFDPS.3

XFDPS.3 (see Table 8) is a variation on the XFDPS.2 model. In this case, the difference exists in the maintenance of resource availability information. In both XFDPS.1 and XFDPS.2, each physical node maintains information about its own resources. XFDPS.3, though, utilizes the approach

Table 7. The XFDPS.2 Control Model

TASK GRAPH CONSTRUCTION:

Who builds the task graph?
Multiple nodes.

What is the nature of the task graph?
Multiple structures each consisting of a subgraph.

Where is the task graph stored?
Multiple nodes.

When is the task graph built?
Piecemeal.

RESOURCE AVAILABILITY INFORMATION:

Who maintains this information?
Each node maintains information about its own resources.

Where is the information maintained?
Separate pieces of information concerning a particular resource type may be kept on differentt nodes.

ALLOCATION OF RESOURCES:

How is concurrency control provided?
Reservations are used prior to a work distribution decision and then allocated by a lock.

PROCESS INITIATION:

How is responsibility distributed?
There is a hierarchy of responsibility.

How is refusal of a request to execute a process by a node handled?
After repeated attempts, the request is abandoned.

PROCESS MONITORING:

What type of interprocess communication is provided?
Unsynchronized communication.

How are task graphs resulting from additional work requests handled?
The new task graph is kept separate.

Table 8. The XFDPS.3 Control Model

TASK GRAPH CONSTRUCTION:

Who builds the task graph?
Multiple nodes.

What is the nature of the task graph?
Multiple structures each consisting of a subgraph.

Where is the task graph stored?
Multiple nodes.

When is the task graph built?
Piecemeal.

RESOURCE AVAILABILITY INFORMATION:

Who maintains this information?
Components for each type of resource.

Where is the information maintained?
Information concerning a particular resource type is kept on a single node.

ALLOCATION OF RESOURCES:

How is concurrency control provided?
Reservations are used prior to a work distribution decision and then allocated by a lock.

PROCESS INITIATION:

How is responsibility distributed?
There is a hierarchy of responsibility.

How is refusal of a request to execute a process by a node handled?
After repeated attempts, the request is abandoned.

PROCESS MONITORING:

What type of interprocess communication is provided?
Unsynchronized communication.

How are task graphs resulting from additional work requests handled?
The new task graph is kept separate.

taken in Medusa which assigns a control component to each type of resource and maintains information concerning a particular type of resource in a single location.

Thus, when resource availability information is required, a resource needs allocation, or a resource needs deallocation, it is only necessary to determine the type of the resource in order to determine the proper control component to perform the desired operation. This is in contrast to XFDPS.1 and XFDPS.2 both of which require a search for the correct component.

SECTION 9

THE EVALUATION OF THE MODELS

9.1 EVALUATION PLAN

As stated earlier in this report, it was planned from the initiation of this survey of control models that it would be followed immediately by an evaluation study of the various models identified or developed. It was also anticipated that this evaluation would cover both the quantitative and qualitative aspects of the various models.

To support the quantitative evaluation of the various forms of system control, a distributed control model simulator is being developed.

9.2 EVALUATION CRITERIA

A number of evaluation criteria have already been identified. The tentative list is summarized in Table 9.

Table 9. Possible Evaluation Criteria for Distributed Control Models

- RESOURCE UTILIZATION
  - Memory Space Utilization
    - By the Control Algorithm
      - Complexity
      - Redundancy
    - By the Control Information
  - Time
    - Local Processing Time
    - Communications Delays
    - Delays in Work Initiation
  - Communication
    - Complexity
    - Quantity
- PERFORMANCE
  - Throughput
  - Response Time
  - Bottlenecks
- SYSTEM FLEXIBILITY
  - Reconfiguration Potential
  - Modularity
    - Logical Complexity
    - Maintainability
    - Problem Partitioning and Algorithm Design
- FAULT-TOLERANCE
  - Detection
  - Recovery
  - Extent to Which Processed Work Can Be Recovered
- PROTECTION
  - Privacy
  - Security

REFERENCES

Akin78 Akin, T. Allen, Flinn, Perry B., Forsyth, Daniel H., "A Prototype for an Advanced Command Language," *Proceedings of the 16th Annual Southeastern Regional ACM Conference* (April, 1978): 96-102.

Ande75 Anderson, George A., and Jensen, E. Douglas., "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," *Computing Surveys* 4 (December, 1975): 197-213.

Bach78 Bachman, Charles, and Canepa, Mike, "The Session Control Layer of an Open System Interconnection," *COMPCON Fall 78* (September, 1978): 150-156.

Balz71 Balzer, R. M., "PORTS - A Method for Dynamic Interprogram Communication and Job Control," *AFIPS Conference Proceedings* 38 (1971 Spring Joint Computer Conference): 485-489.

Brin78 Brinch Hansen, Per, "Distributed Processes: A Concurrent Programming Concept," *Communications of the ACM* 21 (November, 1978): 934-941.

Caba79a Cabanel, J. P., Marouane, M. N., Besbes, R., Sazbon, R. D., and Diarra, A. K., "A Decentralized OS Model for ARAMIS Distributed Computer System," *Proceedings of the First International Conference on Distributed Computing Systems* (October, 1979): 529-535.

Caba79b Cabanel, J. P., Sazbon, R. D., Diarra, A. K., Marouane, M. N., and Besbes, R., "A Decentralized Control Method in a Distributed System," *Proceedings of the First International Conference on Distributed Computing Systems* (October, 1979): 651-659.

Clar80 Clark, David D., and Svobodova, Liba, "Design of Distributed Systems Supporting Local Autonomy," *COMPCON Spring 80* (February, 1980): 438-444.

Cook80 Cook, Robert P., "The STARMOD Distributed Programming System," *COMPCON Fall 80* (September, 1980): 729-735.

Davi79 Davies, D. W., Barber, D. L. A., Price, W. L., and Solomonides, C. M., *Computer Networks and Their Protocols*, John Wiley and Sons, 1979.

Denn78 Denning, Peter J., "Operating Systems Principles for Data Flow Networks," *Computer* (July, 1978): 86-96.

Desj78 desJardins, Richard, and White, George, "ANSI Reference Model for Distributed Systems," *COMPCON Fall 78* (September, 1978): 144-149.

Ensl74 Enslow, Philip H., Jr. (ed.), *Multiprocessors and Parallel Processing*, New York: John Wiley and Sons, 1974.

Ensl78 Enslow, Philip H., Jr., "What is a 'Distributed' Data Processing System?" *Computer* (January, 1978): 13-21.

Farb73 Farber, D. J., Feldman, J., Heinrich, F. R., Hopwood, M. D., Larson, K. C., Loomis, D. C., and Rowe, L. A., "The Distributed Computing System," *COMPCON Spring 73* (February, 1973): 31-34.

Feld79 Feldman, J. A., "High Level Programming for Distributed Computing," Communications of the ACM 22 (June, 1979): 353-368.

Garc79 Garcia-Molina, H., "Performance Comparison of Update Algorithms for Distributed Databases, Crash Recovery in the Centralized Locking Algorithm," Progress Report No. 7, Stanford University, 1979.

Have78 Haverty, J. F., and Rettberg, R. D., "Inter-process Communications for a Server in UNIX," COMPCON Fall 78 (September, 1978): 312-315.

Hoar78 Hoare, C. A. R., "Communicating Sequential Processes," Communications of the ACM 21 (August, 1978): 666-677.

Hopp79 Hopper, K., Kugler, H. J., and Unger, C., "Abstract Machines Modelling Network Control Systems," Operating Systems Review 13 (January, 1979): 10-24.

Jens78 Jensen, E. Douglas., "The Honeywell Experimental Distributed Processor - An Overview," Computer (January, 1978): 28-38.

Kimb76 Kimbleton, Stephen R., and Mandell, Richard L., "A Perspective on Network Operating Systems," AFIPS Conference Proceedings 45 (1976 National Computer Conference): 551-559.

Lein58 Leiner, A. L., and Weinberger, A., "PILOT, the NBS Multicomputer System," Proceedings of the Eastern Joint Computer Conference (1958): 71-75.

Macc80 Maccabe, Aurthur B., and Leblanc, Richard J., "A Language Model for Fully Distributed Systems," COMPCON Fall 80 (September, 1980): 723-728.

Metc76 Metcalfe, R. M., and Boggs, D. R., "Ethernet - Distributed Packet Switching for Local Computer Networks," Communications of the ACM 19 (July, 1976): 395-404.

Morg77 Morgan, Howard L., and Levin, K. Dan, "Optimal Program and Data Locations in Computer Networks," Communications of the ACM 20 (May, 1977): 315-322.

Nels78 Nelson, David L., and Gordon, Robert L., "Computer Cells - A Network Architecture for Data Flow Computing," COMPCON Fall 78 (September, 1978): 296-301.

Oust80 Ousterhout, John K., "Partitioning and Cooperation in a Distributed Multiprocessor Operating System: Medusa," Ph.D. Thesis, Carnegie-Mellon University, April, 1980.

Oust80 Ousterhout, John K., Scelza, Donald A., and Sindhu, Pradeep S., "Medusa: An Experiment in Distributed Operating System Structure," Communications of the ACM 23 (February, 1980): 92-105.

Peeb80 Peebles, Richard, and Dopirak, Thomas, "ADAPT: A Guest System," COMPCON Spring 80 (February, 1980): 445-454.

Ritc78 Ritchie, D. M., and Thompson, K., "The UNIX Time-Sharing System," The Bell System Technical Journal 57 (July-August, 1978): 1905-1929.

Sapo80 Saponas, Timothy G., and Crews, Phillip L., "A Model for Decentralized Control in a Fully Distributed Processing System," COMPCON Fall 80 (September, 1980): 307-312.

Smit79 Smith, Reid G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," Proceedings of the 1st International Conference on Distributed Computing (October, 1979): 185-192.

Smit80 Smith, Reid G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," IEEE Transactions on Computers C-29 (December, 1980): 1104-1113.

Suns77 Sunshine, Carl, "Interprocess Communication Extensions for the UNIX Operating System: I. Design Considerations," Rand Technical Report R-2064/1-AF, June 1977.

Thom78 Thomas, Robert H., Schantz, Richard E., and Forsdick, Harry C., "Network Operating Systems," Bolt Beranek and Newman Report No. 3796 (March, 1978).

Ward80 Ward, Stephen A., "TRIX: A Network-Oriented Operating System," COMPCON Spring 80 (February, 1980): 344-349.

Zuck77 Zucker, Steven, "Interprocess Communication Extensions for the UNIX Operating System: II. Implementation," Rand Technical Report R-2064/2-AF, June, 1977.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

LMED

-8